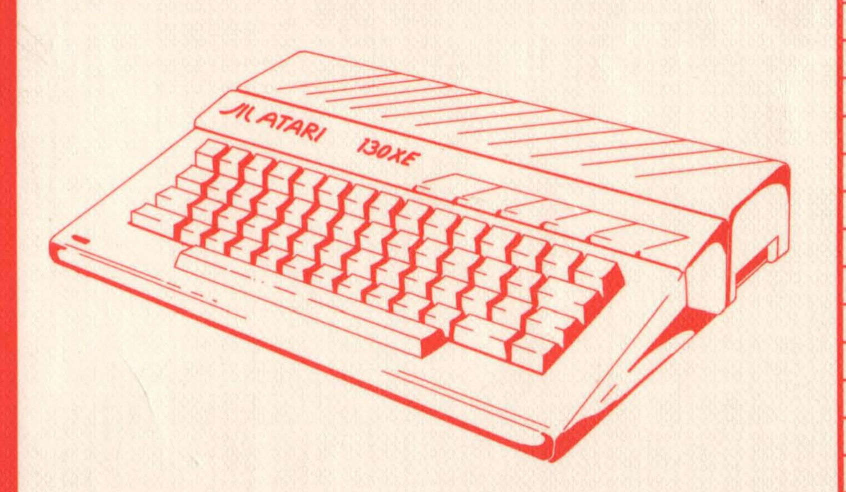
SZCZEPANOWSKI

ATARI®

130 XE / 600 XL / 800 XL



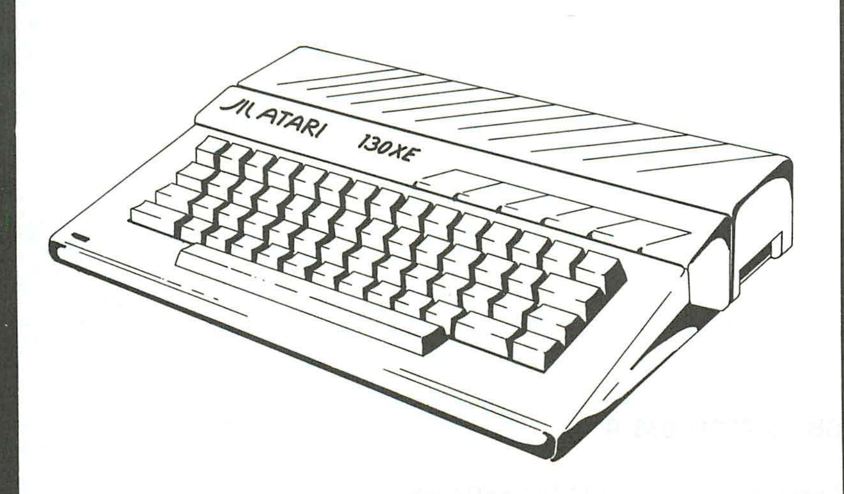
FÜR EINSTEIGER

EIN DATA BECKER BUCH

SZCZEPANOWSKI

ATARI®

130 XE / 600 XL / 800 XL



FÜR EINSTEIGER

EIN DATA BECKER BUCH

ISBN 3-89011-033-9

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

KAPITEL 1:

DIE TASTATUR

KAPITEL 2:

DER ERSTE BEFEHL

KAPITEL 3:

DAS ERSTE PROGRAMM

KAPITEL 4:

BASIC-EINFÜHRUNG

KAPITEL 5:

FARBE UND GRAFIK

KAPITEL 6:

TONERZEUGUNG

KAPITEL 7:

NOCH MEHR BEFEHLE

ANHANG

KAPITEL 1: DIE TASTATUR



In diesem Kapitel lernen Sie die Handhabung der 57 Tasten Ihres ATARI. Mit Hilfe dieser Tasten werden Sie alle verborgenen Möglichkeiten Ihres Heimcomputers erschließen. Sei es die Eingabe von Texten, das Erstellen von Bildern (Grafiken), das Abschicken von Anweisungen oder die Berechnung mathematischer Probleme, all dies wird für Sie nach diesem Kapitel kein Neuland mehr sein. Wenn Sie einmal alle Tasten beherrschen, wird der ATARI Ihnen ein folgsamer Partner bei der Lösung vieler Probleme sein.

Das Beherrschen der Tastatur dient nicht nur Computer-Freaks, die einmal jedes Bit des Rechners mit dem Vornamen kennen möchten, sondern auch denen, die ihren Computer nur mit den reichhaltig angebotenen Fertigprogrammen "füttern" wollen. Jemand, der sich zu den letztgenannten Anwendern zählt, darf bei einer Meldung eines Standardprogramms wie z.B. "DRUECKEN SIE DIE RETURN-TASTE ZUM AUSDRUCK DES BILDSCHIRMINHALTS" nicht verzweifelt in seinem Handbuch wühlen.

Kurzum sollte jeder, der auch nur gelegentlich an seinem Gerät arbeitet, mit der Tastatur vertraut sein. Der Vater eines computerfaszinierten Sohnes z.B. sollte zumindest wissen, wie das beste Spielprogramm des Sohnemanns geladen wird, wenn der gerade außer Haus ist.

Doch keine Angst, Sie müssen sich nicht vorher zu einem Schreibmaschinenkursus an der Volkshochschule anmelden. Die meisten auch noch so erfahrenen Freizeitprogrammierer arbeiten mit dem "Zweifinger-Suchsystem". Wer eine gewisse Zeit mit der Tastatur vertraut ist, wird sich bald darüber wundern, wie flink er über die Tasten saust.

In diesem Kapitel werden die Hilfstasten besonders ausführlich beschrieben. Diese Tasten sind sehr wichtig, da z.B. Texte bearbeitet, Programme unterbrochen und Befehle an den Rechner übermittelt werden können.

ALLGEMEINES ZUR TASTATUR

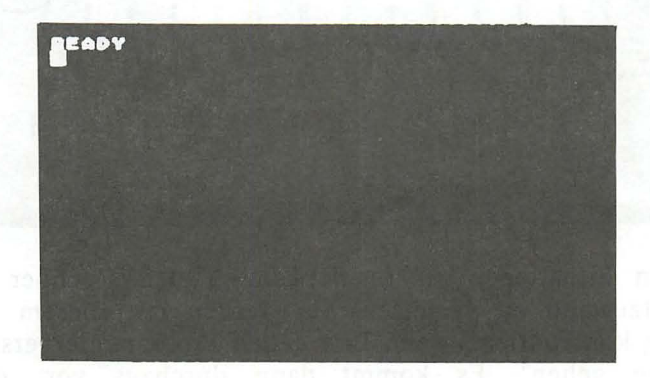
Auf den ersten Blick erweckt die Tastatur des ATARI den Eindruck einer üblichen Schreibmaschinentastatur. Doch wenn Sie genauer hinsehen, werden Sie leichte Abweichungen feststellen

> Die Buchstaben 'Y' und 'Z' sind gemäß der amerikanischen ASCII-Norm vertauscht.
>
> Die Tastatur weist auf den ersten Blick keine Umlaute (ö, ä, ü) und kein scharfes 's' (ß) vor, da diese Zeichen im amerikanischen Zeichensatz nicht enthalten sind.
>
> Es gibt zusätzliche Tasten (Hilfstasten), deren Funktion später erläutert wird.

Sie sollten keine Experimente mit der Tastatur machen, bevor Sie nicht mit deren Funktionen vertraut sind. Zwar können durch Fehlbedienung keine Rauchwolken aus dem Rechner aufsteigen,

doch Sie können durch Mißerfolge verblüfft werden. Warten Sie also geduldig auf die praktische Einweisung auf den nächsten Seiten.

LOS GEHT'S



Nun wird es Zeit, die schlafende Technik in Ihrem ATARI zum Leben zu erwecken. Schalten Sie dazu den Rechner ein. Sie finden einen Schalter hinten links am Rechner. Nach jedem Einschalten erscheint eine Meldung, die besagt, daß der Rechner nun bereit ist, Ihren Kommandos zu folgen. Dieses 'READY' signalisiert, daß das Betriebssystem nun Kommandos erwartet. Der Rechner steht Ihnen jetzt zur Verfügung.

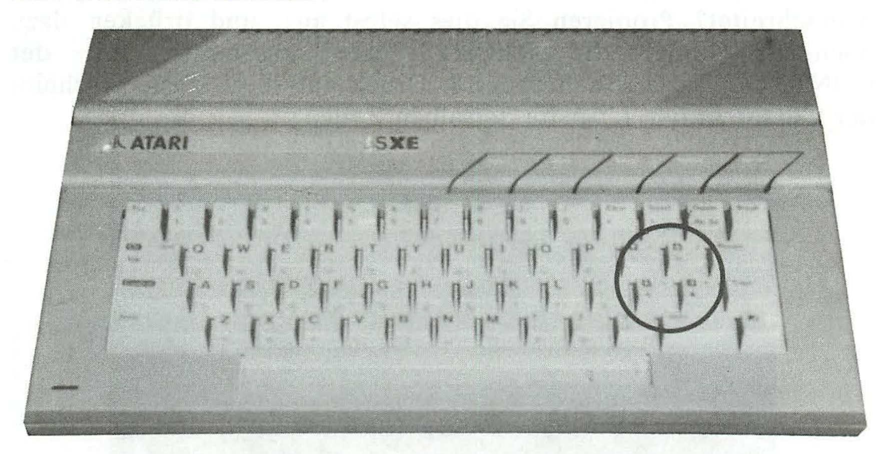
Nun zu dem kleinen Quadrat unterhalb des 'READY'. Dies ist eine Orientierungshilfe auf dem Bildschirm. Angenommen, Sie möchten etwas auf dem Bildschirm schreiben, dann ist es notwendig, genau zu wissen, wo das einzugebende Zeichen erscheinen wird. Diese Markierung, die man in der Fachsprache CURSOR ("körsa" gesprochen) nennt, hilft Ihnen also, sich am Bildschirm zurechtzufinden.

RECHNER-RESET

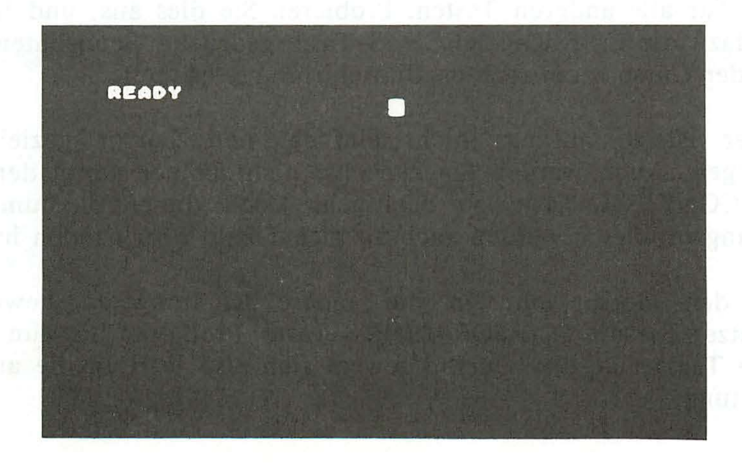


In vielen Situationen ist es notwendig, den Rechner in den Einschaltzustand zu versetzen. Sie werden mit diesem Problem erstmalig konfrontiert, wenn Ihre ersten Programmierversuche "in die Hose gehen". Es kommt dann durchaus vor, daß der Programmierer nicht an den Ausgang des Programms gedacht hat, und das Programm nicht im Traum daran denkt, seinen Lauf abzubrechen bzw. zu beenden. Bei vielen anderen Computern muß dieser dazu aus- und wieder eingeschaltet werden. Dies hat aber eine unnötige Belastung der hochwertigen Elektronik des Rechners zur Folge. Um dies zu vermeiden, kann der ATARI mit einem Tastendruck in den Einschaltzustand versetzt werden. Diese Taste befindet sich ganz rechts oben an dem Rechner. Hier sind fünf Spezialtasten angeordnet, von denen die mit der Bezeichnung "Reset" zum gewünschten Erfolg führt. Betätigen Sie nun diese Taste. Sie bemerken, daß das gleiche Bild erscheint, das auch dem Einschalten des Rechners folgt. BASIC-Programme jedoch werden dabei nicht gelöscht.

DIE CURSOR-TASTEN



Diese Tasten befinden sich auf der rechten Hälfte der Tastatur. Die Pfeile auf diesen Tasten entsprechen der Richtung, in die sich der Cursor auf dem Bildschirm bewegen soll. Diese Tasten haben jedoch drei Funktionen. Soll der Cursor bewegt werden, so ist es notwendig, zusätzlich die Taste 'CONTROL' zu drücken. Dazu halten Sie diese Taste gedrückt, um anschließend den Cursor zu bewegen. Betätigen Sie nun einmal die RECHTS-Taste 20mal. Vergessen Sie nicht, die CONTROL-Taste gedrückt zu halten! Der Cursor wandert 20 Stellen nach rechts und verweilt wieder geduldig in der Bildschirmmitte.



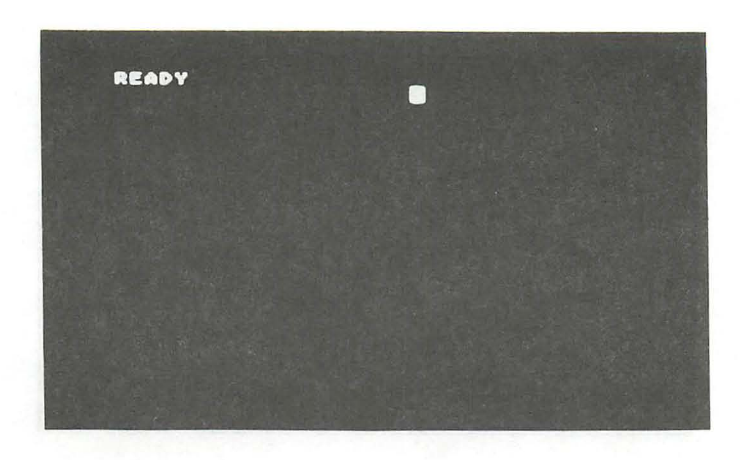
Was aber, wenn der Cursor den rechten Rand dieser Zeile überschreitet? Probieren Sie dies selbst aus, und drücken dazu nochmals 18mal die RECHTS-Taste, zusammen mit der CONTROL-Taste. Nach dem 18. Druck auf diese Taste erscheint der Cursor in der 1. Spalte derselben Zeile.



Es ist doch recht umständlich, 20mal auf die Cursortaste zu hämmern, um in die Bildschirmmitte zu gelangen. Diese Prozedur können Sie etwas vereinfachen. Wenn Sie die Cursortaste gedrückt halten, so wandert der Cursor selbständig nach rechts. Dies gilt auch für alle anderen Tasten. Probieren Sie dies aus, und halten Sie dazu die CURSOR-RECHTS-Taste gedrückt. Beobachten Sie, wie der Cursor zum rechten Bildschirmrand sprintet.

Sicher ist es anfangs nicht einfach, den Cursor gezielt zu bewegen. Auch werden Sie zunächst nicht immer daran denken, die CONTROL-Taste zu betätigen. Doch dieses Stadium der Anfangsprobleme werden auch Sie sicher bald überwunden haben.

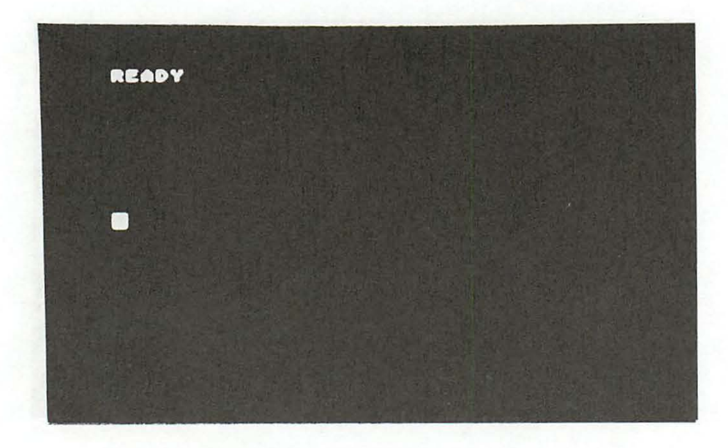
Um den Cursor nun in die andere Richtung zu bewegen, benutzen Sie die CURSOR-LINKS-Taste. Probieren Sie nun auch diese Taste aus. Der Cursor bewegt sich also jetzt in die andere Richtung.



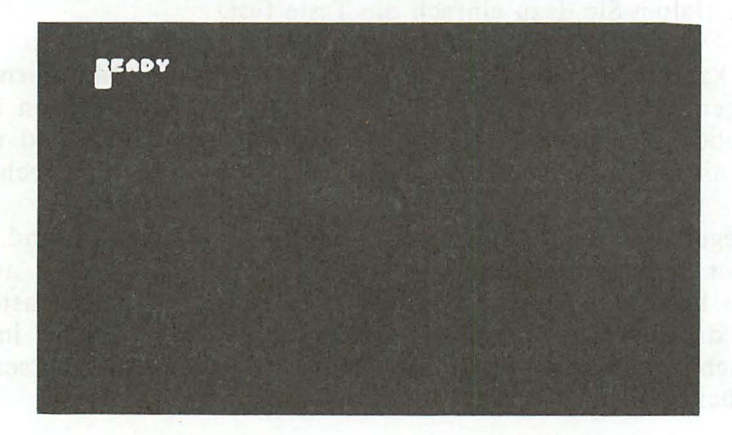
Selbstverständlich bewegt sich der Cursor auch selbständig nach links, ohne daß immer wieder auf die Taste gedrückt werden muß. Halten Sie dazu einfach die Taste fest.

Wie kann man nun den Cursor in die mittlere Bildschirmzeile bringen? Die Cursortasten OBEN und UNTEN ermöglichen Ihnen die ebenso schnelle Bewegung des Cursors nach oben und unten wie mit den bisher beschriebenen Tasten nach links und rechts.

Bewegen Sie nun den Cursor zum linken Bildschirmrand. Der Cursor wartet wiederum geduldig, bis Sie ihn an eine andere Stelle bewegen. Peilen Sie nun die CURSOR-UNTEN-Taste an, und drücken Sie diese Taste solange, bis der Cursor in der Bildschirmmitte ist. Auch hier können Sie dazu die Cursortaste gedrückt halten.



Versuchen Sie nun, den Cursor in die 3. Bildschirmzeile zu bewegen. Benutzen Sie dazu die CURSOR-OBEN-Taste.



Auch hier wandert der Cursor selbständig Zeile für Zeile nach oben oder nach unten, wenn Sie die Taste gedrückt halten.

Um ein Gefühl für die Cursor-Tasten zu erlangen, versuchen Sie einmal, mit dem Cursor ein unsichtbares Quadrat zu zeichnen. D.h. Sie bewegen den Cursor nach rechts - unten - links - oben oder anders herum nach links - unten - rechts - oben. Das

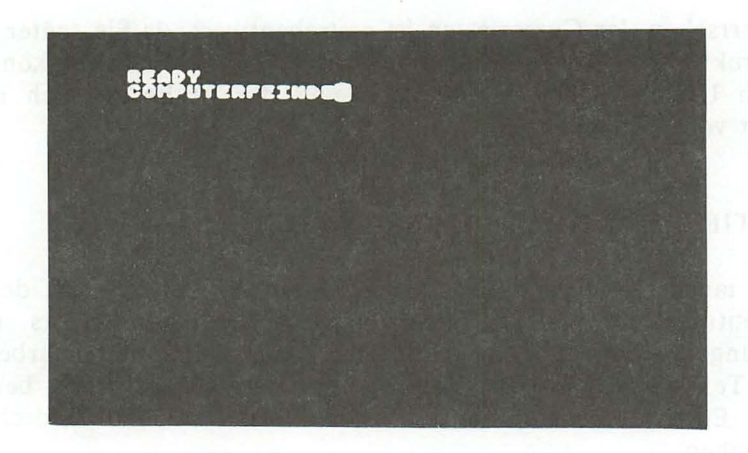
Beherrschen der Cursortasten ist erstrebenswert, da Sie später z.B. Korrekturen in einem Programm blitzschnell vornehmen können. Doch Übung macht den Meister, und der ist bekanntlich noch nicht vom Himmel gefallen.

EDITIEREN MIT DEN CURSOR-TASTEN

Hier taucht ein Fremdwort aus der Datenverarbeitung auf, dessen Bedeutung dem einen oder anderen Leser, besonders dem Anfänger, sicher nicht bekannt ist. Editieren bedeutet Bearbeiten von Texten, z.B. Texte erstellen und ändern. Man spricht bereits vom Editieren, wenn Sie ein paar Worte auf den Bildschirm schreiben.

Mit Ihrem ATARI ist das Editieren recht unproblematisch. Der Grund dafür ist der komfortable Bildschirmeditor. Sie können den Cursor an jede beliebige Stelle des Bildschirms bewegen und dort Texte schreiben, ändern usw. Der Vorteil dieses Bildschirmeditors wird Ihnen erst deutlich, wenn Sie die ersten BASIC-Programme schreiben. Dann nämlich können Sie den Cursor auf eine beliebige, auf dem Bildschirm befindliche Programmzeile bewegen und diese ändern. Doch dazu an später mehr.

Sicher wollen Sie gleich diesen Editor ausprobieren. Geben Sie dazu unterhalb der Meldung 'READY' das Wort 'COMPUTERFEINDE' ein.



Sie werden feststellen, daß der Cursor mit jedem eingegebenen Buchstaben selbständig eine Stelle nach rechts wandert. Nach Eingabe des Wortes verweilt der Cursor hinter dem letzten Buchstaben. An jeder beliebigen Stelle dieses Wortes können wir nun Änderungen vornehmen, da der Cursor weiterhin beliebig bewegt werden kann. Wollen Sie z.B. aus dem 'COMPUTERFEINDE' das Wort 'COMPUTERFREUNDE' machen, so bewegen Sie den Cursor zunächst zum ersten zu ändernden Buchstaben.



Nun können Sie die alten Buchstaben einfach überschreiben. Geben Sie dazu den restlichen Text 'REUND' ein.



Das Wort ist nun durch einfaches Überschreiben geändert worden. Sie können grundsätzlich jedes Zeichen, das sich auf dem Bildschirm befindet, ändern, indem Sie den Cursor auf diese Stelle bewegen und ein anderes Zeichen eingeben.

LÖSCHEN DES BILDSCHIRMS

Stellen Sie einmal fest, daß das, was sich auf dem Bildschirm befindet, eigentlich alles unbrauchbares Kauderwelsch ist, so wischen Sie einfach alles weg. HALT!! Nicht mit dem Fensterleder, damit können Sie vielleicht den Staub vom Bildschirm wischen, nicht jedoch die eingetippten und somit im Rechner gespeicherten Daten des Bildschirms. Schließlich ist es für einen Rechner, der ca. 1 Million Befehle pro Sekunde abarbeiten kann, ein Kinderspiel, "mal eben" die 1000 Zeichen auf dem Bildschirm mit einem Tastendruck zu löschen. Diese Taste heißt 'CLEAR', befindet sich links von der Taste 'INSERT' und muß ebenfalls zusammen mit CONTROL oder SHIFT betätigt werden. Probieren Sie dies nun aus, so werden alle Zeichen auf dem Bildschirm gelöscht, und der Cursor befindet sich in der oberen linken Bildschirmecke.

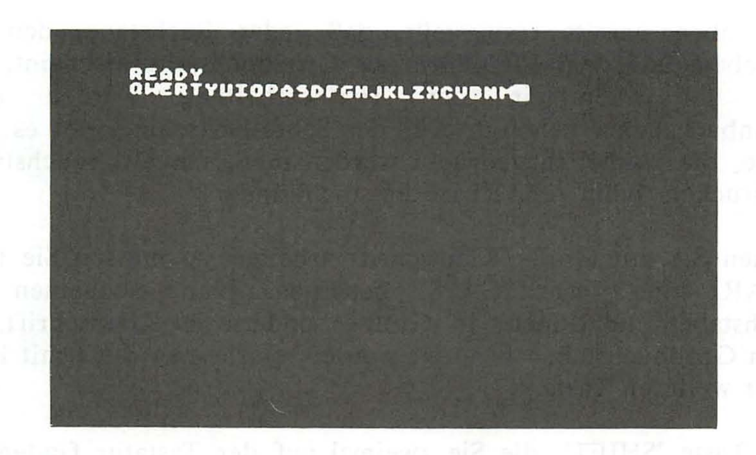
DIE TASTEN MIT BUCHSTABEN



Wie bereits erwähnt, gleicht die Anordnung dieser Tasten bis auf kleine Abweichungen der einer Schreibmaschine. Konzentrieren wir uns zunächst auf diese Buchstaben. Wenn Sie irgendeine dieser Tasten drücken, so erscheint der entsprechende Kleinbuchstabe an der Cursor-Position auf dem Bildschirm. Bevor wir diese Tasten einsetzen, löschen Sie bitte den Bildschirm mit der Tastenkombination CONTRL + CLEAR.

Nun haben wir einen "sauberen" Bildschirm, auf dem wir editieren können. Bitte geben Sie nun keine Unmengen von Texten ein, in der Hoffnung daß Ihr Computer alles, zu jeder Zeit abrufbereit in seinen Speicher aufnimmt. Ganz so einfach ist es nicht. Das, was Sie eingeben, wird lediglich im flüchtigen Bildschirmspeicher festgehalten. Wie man eingegebene Texte abspeichern kann, erfahren Sie in unserer BASIC-Einführung.

Schreiben Sie jetzt einmal alle Buchstaben auf den Bildschirm, vielleicht sogar in alphabetischer Reihenfolge. Sie werden feststellen, wie schwer doch zu Anfang der ein oder andere Buchstabe zu finden ist, wenn man nicht gerade Sekretärin ist. Auf dem Bildschirm befinden sich nun alle verfügbaren Großbuchstaben.



Löschen Sie wieder den Bildschirm und geben nochmals alle Buchstaben ein. Doch nun nicht in alphabetischer Reihenfolge, sondern wie sie auf der Tastatur angeordnet sind. Also zunächst die obere Reihe (Q bis P), dann die mittlere Reihe (A bis L) und schließlich die untere Reihe (Z bis M). Sicher fällt Ihnen diese Reihenfolge viel leichter.

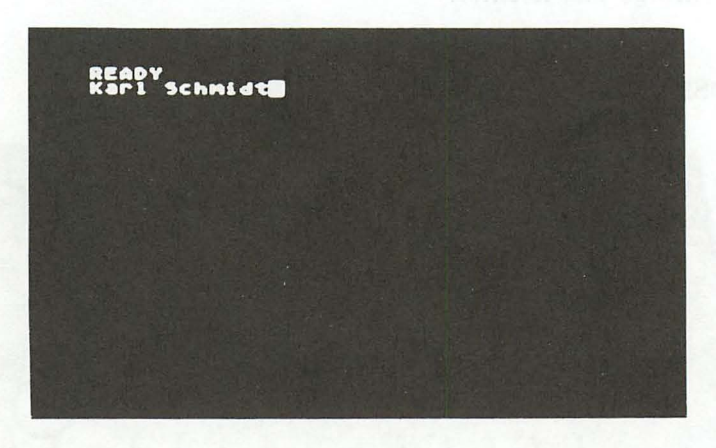
GROSS-/KLEINSCHRIFT



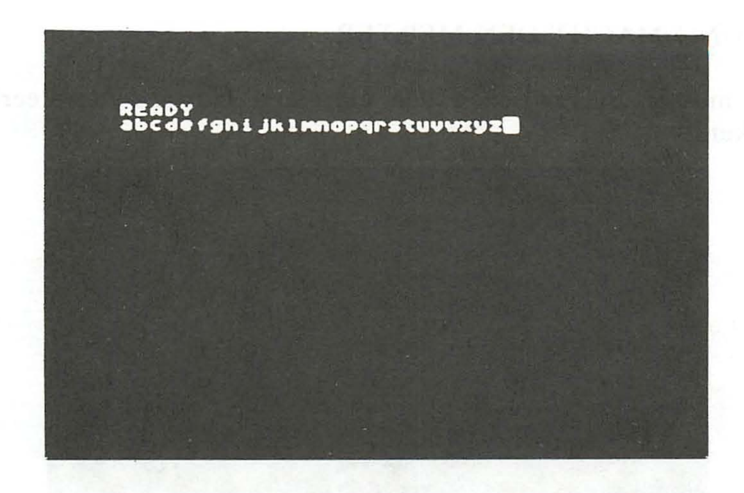
Sie haben bereits festgestellt, daß jeder Buchstabe, den Sie eingeben, auf dem Bildschirm als Großbuchstabe erscheint. Oft und vor allem in der Textbearbeitung werden auch Kleinbuchstaben benötigt. Auf der Schreibmaschine gibt es eine Taste, die zusätzlich gedrückt werden muß, um Großbuchstaben zu drucken. Beim ATARI ist dies nicht anders.

Wollen Sie mit Groß-/Kleinschrift arbeiten, so müssen Sie beim ATARI die Taste 'CAPS' betätigen. Nun erscheinen die Buchstaben nicht mehr in Groß-, sondern in Kleinschrift. Da auch Großbuchstaben benötigt werden, erscheinen diese mit Hilfe einer weiteren Taste.

Die Taste 'SHIFT', die Sie zweimal auf der Tastatur finden, ist für die Großbuchstaben zuständig. Halten Sie also diese Taste fest, wenn Sie Großbuchstaben wünschen. Schreiben Sie nun einmal den Namen "Karl Schmidt" auf dem Bildschirm, nachdem Sie mit der Taste 'CAPS' die Groß-/Kleinschrift aktiviert haben. Die Großbuchstaben werden wie bereits bekannt mit der SHIFT-Taste eingegeben.



Wir haben bereits alle Großbuchstaben auf dem Bildschirm geschrieben. Lassen Sie uns nun auch die Kleinbuchstaben in alphabetischer Reihenfolge eingeben.



Drücken Sie die Taste 'CAPS' ein weiteres Mal, so können wieder nur Großbuchstaben eingegeben werden.

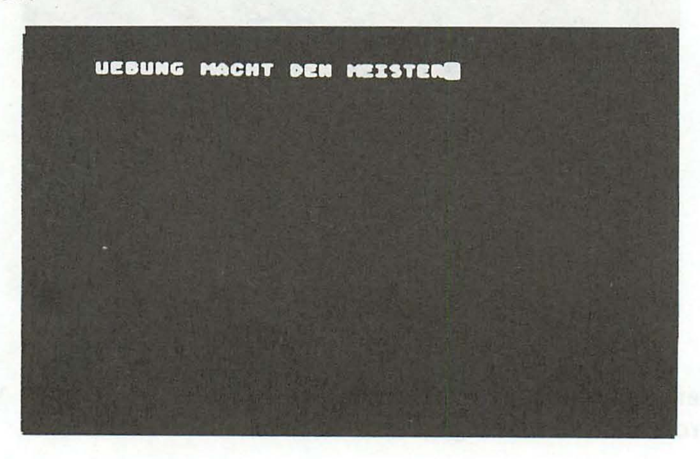
DIE LEERTASTE



Diese Taste finden Sie ebenfalls bei jeder Schreibmaschine wieder. Auch bei Ihrem Computer brauchen Sie nicht darauf zu verzichten, Leerzeichen zwischen die Wörter zu setzen. Befindet sich an der Stelle, an der ein Leerzeichen erscheinen soll, bereits ein anderes Zeichen, so wird dieses natürlich gelöscht.
Löschen Sie den Bildschirm und geben den folgenden Satz ein:

UEBUNG MACHT DEN MEISTER

Hier müssen Sie zwischen den einzelnen Wörtern die Leertaste drücken.



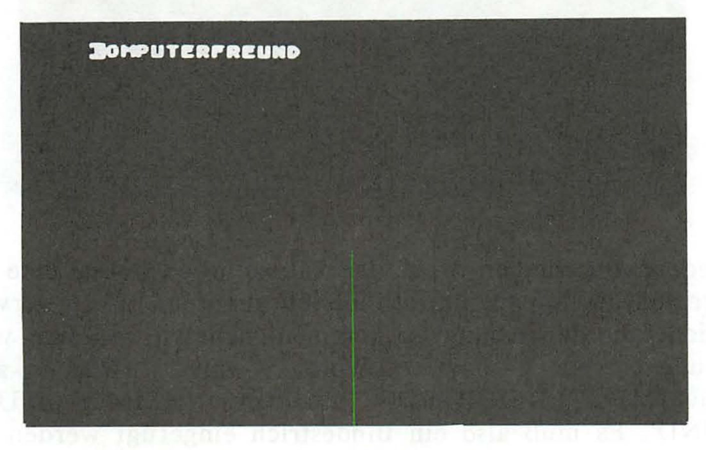
DIE INSERT-TASTE



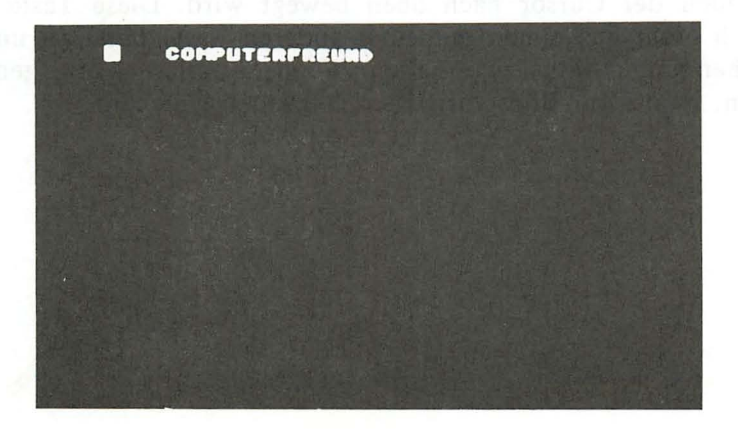
Das Überschreiben von Zeichen ist nicht die einzige Editierhilfe, die der ATARI bietet. Eine weitere Editierhilfe ist das Einfügen. Hierfür besitzt der ATARI eine spezielle Taste. Zum Einfügen muß der Cursor zunächst auf den Buchstaben positioniert werden, vor den ein weiterer Buchstabe eingefügt werden soll. Dann wird

die INSERT-Taste (zusammen mit CONTROL) solange betätigt, bis der gewünschte Platz eingefügt ist. Der einzufügende Text kann dann eingegeben werden. Ein Beispiel soll dies verdeutlichen.

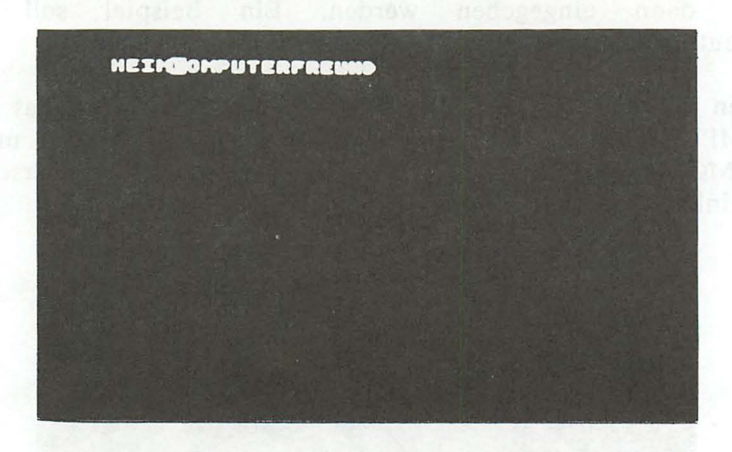
Geben Sie zunächst nach dem Löschen des Bildschirms das Wort 'COMPUTERFREUND' ein und ändern es anschließend um in 'HEIMCOMPUTERFREUND'. Bewegen Sie dazu den Cursor an den linken Rand des Wortes 'COMPUTERFREUND'.



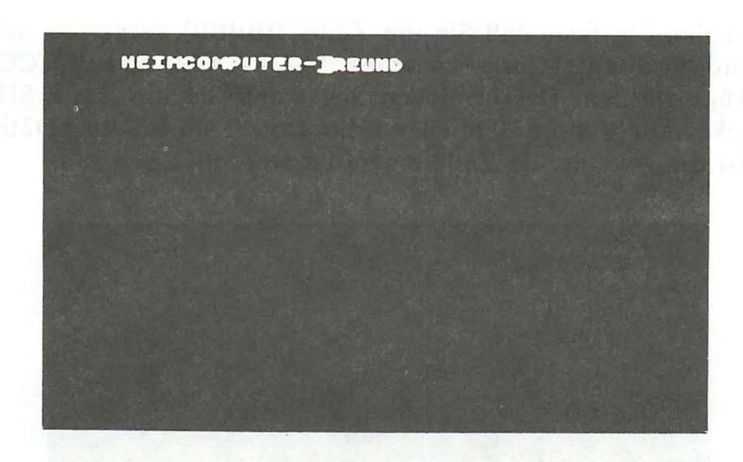
Der Cursor befindet sich nun auf dem Buchstaben 'C', vor den noch die Silbe 'HEIM' eingegeben werden muß. Um Platz für diese vier Buchstaben zu schaffen, halten Sie die CONTROL-Taste gedrückt und betätigen viermal die INSERT-Taste.



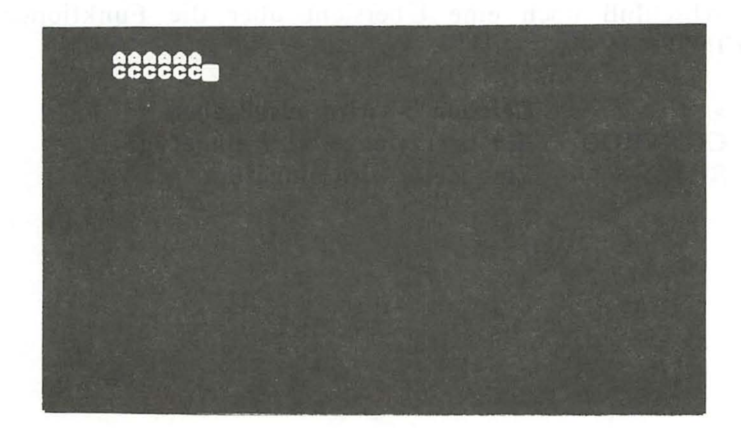
Nun können Sie die gewünschten Buchstaben eingeben. Drücken Sie nacheinander die Buchstaben 'H', 'E', 'I' und 'M'.



Mit jedem Buchstaben wird der Cursor wieder um eine Stelle nach rechts geschoben, um schließlich auf dem 'C' zu verweilen. Um nicht aus der Übung zu kommen, nehmen wir eine weitere Änderung an dem Wort vor. Wir ändern 'HEIMCOMPUTERFREUND' um in 'HEIMCOMPUTER-FREUND'. Es muß also ein Bindestrich eingefügt werden. Dazu bewegen wir wieder den Cursor auf das Zeichen, vor dem eingefügt werden soll, und drücken einmal die INSERT-Taste, zusammen mit CONTROL. Anschließend geben wir den Bindestrich ein. Dieses Zeichen befindet sich auf der Taste, mit der auch der Cursor nach oben bewegt wird. Diese Taste wird jedoch nicht zusammen mit einer anderen Taste betätigt, um das Zeichen unten links zu erhalten. Wenn Sie alles richtig gemacht haben, müßte Ihr Bildschirm dem folgenden gleichen:



Die Taste INSERT hat eine weitere Funktion. Mit ihr können nicht nur Zeichen, sondern auch ganze Zeilen eingefügt werden. Dies wird erreicht, wenn Sie diese Taste nicht zusammen mit CONTROL, sondern zusammen mit SHIFT betätigen. Das folgende Beispiel schafft auch hier Klarheit. Löschen Sie den Bildschirm und geben zwei Zeilen Text ein. Um die zweite Zeile zu erreichen, bewegen Sie den Cursor nach Eingabe der ersten Zeile zunächst nach links und anschließend eine Zeile nach unten.



Nun stellen Sie fest, daß Sie die Zeile 'BBBBB' vergessen haben. Es muß also eine Zeile zwischen 'AAAAAA' und 'CCCCCC' eingefügt werden. Das erreichen Sie, wenn Sie die Taste SHIFT gedrückt halten, um anschließend die Taste INSERT zu betätigen. Nun ist die gewünschte Zeile eingefügt worden.



Nun können Sie die Zeile 'BBBBBB' eingeben. Zum Einfügen von Zeilen sei noch bemerkt, daß der Cursor auf die Zeile gesetzt werden muß, VOR der eine weitere Zeile eingefügt werden soll. Zum Abschluß noch eine Übersicht über die Funktionen der Taste INSERT:

>	**Zeichen '>' wird ausgegeben**
CONTROL >	**Ein Leerzeichen wird eingefügt**
SHIFT >	**Eine Zeile wird eingefügt**

DIE DELETE/BACKSPACE-TASTE



Die Abkürzung dieser Taste sagt bereits viel über die Funktion aus. DELETE z.B. bedeutet löschen. Mit der Taste DELETE BACKSPACE (beim 130 XE beschriftet als *Deltete Bk Sp*) kann das Zeichen links vom Cursor gelöscht werden. Dies ist besonders hilfreich beim Korrigieren von Tippfehlern. Ein Druck auf diese Taste und das falsch getippte Zeichen ist wieder gelöscht.

Löschen Sie nun den Bildschirm und tippen den folgenden Satz ein:

TIPPFEHLER SIND UNVERZEIHLICJ

Bei der Eingabe des letzten Zeichens ist ein Fehler unterlaufen.



Tipp-Ex ist hier nicht notwendig. Auf Tastendruck wird der zuletzt eingegebene Buchstabe gelöscht. Drücken Sie dazu die Taste DELETE BACKSPACE. Die Funktion BACKSPACE wird aktiv und löscht das zuletzt eingegebene Zeichen.



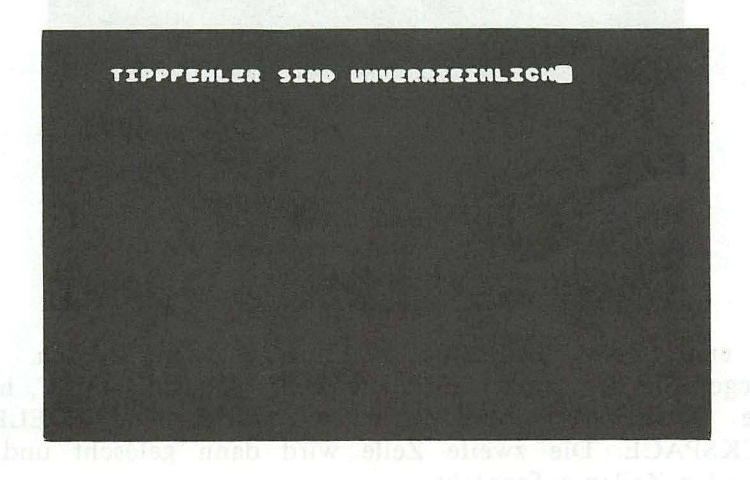
Nun können Sie hier den richtigen Buchstaben einsetzen, und der Tippfehler ist vergessen.



Doch dieses Beispiel zeigt nicht den überragenden Vorteil dieser Taste. Spielen wir ein weiteres Beispiel durch. Geben Sie, nachdem Sie den Bildschirm gelöscht haben, den folgenden Satz ein:

TIPPFEHLER SIND UNVERRZEIHLICH

Auch hier hat sich ein Tippfehler eingeschlichen: Zwei 'R' sind hier nicht angebracht.

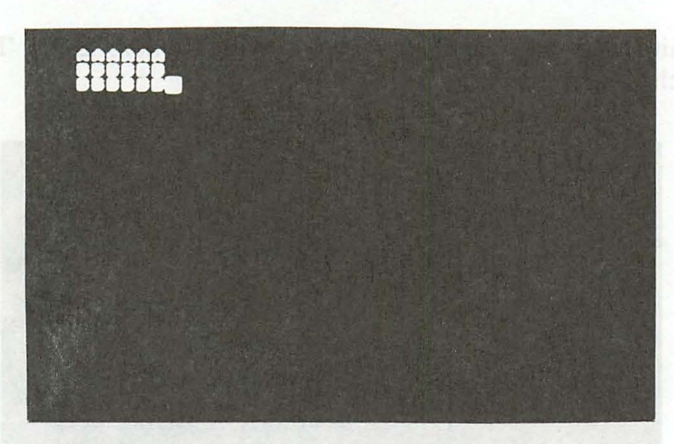


Es ist nicht notwendig, das letzte Wort ab dem Tippfehler nochmal zu schreiben. Die Taste DELETE BACKSPACE hat eine weitere Funktion, nämlich das DELETE. Dazu drücken Sie zusätzlich die Taste 'CONTROL'. Diese Taste löscht nicht das Zeichen links, sondern das Zeichen unter dem Cursor und rückt die Zeichen rechts vom Cursor auf. Das ist nicht einfach zu erklären, sehen Sie darum selbst:

Bewegen Sie den Cursor auf das zu löschende 'R' des fehlerhaften Satzes und drücken die Tasten 'CONTROL' mit '+'. Der Erfolg ist, daß das 'R' unter dem Cursor gelöscht und die anderen Zeichen nachgerückt wurden.

Auch die DELETE/BACKSPACE-Taste hat noch eine dritte Funktion. Mit ihr können auch ganze Zeilen gelöscht werden. Dies wird erreicht, wenn Sie zusammen mit SHIFT gedrückt wird. Auch hier soll ein Beispiel diese Funktion verdeutlichen.

Geben Sie nun die drei folgenden Zeilen ein. Um zur jeweils nächsten Zeile zu gelangen, benutzen Sie die Taste 'BREAK'.



Die erste Zeile 'BBBBBB' soll nun gelöscht werden. Dazu bewegen Sie den Cursor auf diese Zeile, drücken SHIFT, halten diese Taste fest und drücken anschließend DELETE/ BACKSPACE. Die zweite Zeile wird dann gelöscht und alle folgenden Zeilen aufgerückt.

Zum Abschluß auch hier eine Übersicht sämtlicher Funktionen dieser Taste:

DELETE/BACKSPACE	**löscht das Zeichen links vom Cursor**
CONTROL DELETE/BACKSPACE	**löscht das Zeichen unter dem Cursor und rückt die anderen Zeichen nach**
SHIFT DELETE/BACKSPACE	**löscht die Zeile, auf der sich der Cursor befindet und rückt die anderen Zeilen nach**

INVERSE ZEICHENDARSTELLUNG



In der Fachsprache spricht man von der Betriebsart REVERSE. Die deutsche Übersetzung für 'REVERSE' ("rivörs" gesprochen) ist sinngemäß 'UMGEKEHRT', auch invers genannt. Doch was sind 'umgekehrte' Zeichen? Sicher haben Sie schon einmal Negative von Schwarz/Weiß-Fotos betrachtet und festgestellt, daß die Farben vertauscht sind. Ein dunkler Hintergrund des Fotos ist auf dem Negativ hell. Wenn Sie die Zeichen auf dem Bildschirm genau betrachten, so merken Sie, daß sie aus einzelnen Punkten zusammengesetzt sind. Alle Zeichen werden in einer Matrix von 8

mal 8 Bildschirmpunkten dargestellt. Der Buchstabe 'B' z.B. sieht dann etwa so aus:

```
OOOOO
OO  OO
OOOOO
OO  OO
OO  OO
OOOOO
```

Wenn dieses Zeichen nun invers dargestellt wird, werden alle gesetzten Punkte der 8-mal 8-Matrix gelöscht und umgekehrt. Das 'B' sieht demnach dann so aus:

```
OOOOOOOO
O     OO
O  OO  O
O     OO
O  OO  O
O  OO  O
O     OO
OOOOOOOO
```

Wenn Sie schon einmal zufällig die untere rechte Taste des ATARI betätigt haben, sind Sie unfreiwillig mit den inversen Zeichen konfrontiert worden. Die inverse Zeichendarstellung wird mit dieser Taste eingeschaltet.

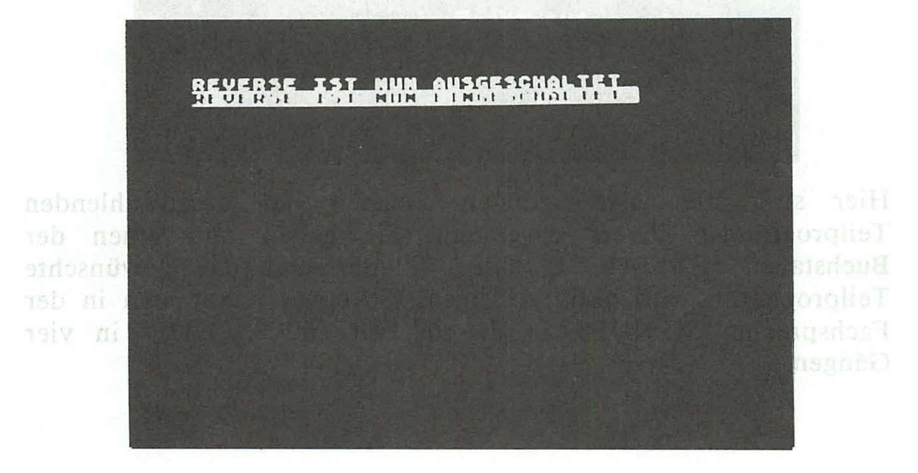
Löschen Sie nun den Bildschirm und schreiben in die erste Zeile den Text

REVERSE IST NUN AUSGESCHALTET



Nachdem Sie den Cursor zum Anfang der zweiten Zeile befördert haben, drücken Sie bitte die INVERS-Taste (unten rechts an der Tastatur). Schreiben Sie danach den folgenden Text auf dem Bildschirm:

REVERSE IST NUN EINGESCHALTET



Jetzt sehen Sie bestens, was inverse Zeichendarstellung bedeutet. Natürlich muß dies auch wieder auszuschalten sein. Dazu drücken Sie nochmals die INVERS-Taste. Nun können Sie in der gewohnten Art und Weise weiterschreiben.

Wozu benötigt man diese ungewohnte Zeichendarstellung? Eine berechtigte Frage, die jedoch leicht zu beantworten ist. Wenn Sie schon einmal Programme aus dem Handel gestartet und bedient haben, werden Sie sicher hier und da invers Zeichen erkannt haben. Z.B. wird die letzte Bildschirmzeile gerne dazu benutzt, den Benutzer des Programms auf Fehler hinzuweisen. Diese Fehlermeldungen werden dazu invers ausgegeben, damit sie sofort ins Auge fallen. Eine weitere Möglichkeit ist, die ersten ein oder zwei Buchstaben von aufgelisteten Teilprogrammen invers darzustellen. Der Benutzer wählt dann das entsprechende Teilprogramm aus, indem er die zugehörigen inversen Zeichen eingibt. Ein Beispiel:



Hier sind die jeweils ersten Zeichen der auszuwählenden Teilprogramme invers dargestellt. Sie geben nun einen der Buchstaben 'E', 'A', 'L' oder 'S' ein und das gewünschte Teilprogramm wird aktiv. Übrigens: So etwas nennt man in der Fachsprache "MENUE", in diesem Fall ein "MENUE" in vier Gängen.

TABULATOREN



Mit dieser Taste können Sie die Tabulatorpositionen anspringen, löschen oder setzen. Nach dem Einschalten sind die Tabulatoren bereits wie folgt gesetzt ('x' gibt die Tabulatorposition an).

```
         1         2         3
12345678901234567890123456789012345678
-----x-------x-------x-------x-------x
```

Wenn Sie die TAB-Taste betätigen, springt der Cursor zur nächsten Tabulatorposition. Diese Tabulatoren lassen sich mit der Tastenkombination CONTROL-TAB löschen. Probieren Sie dies nun aus. Löschen Sie dazu den Bildschirm und gehen wie folgt vor: Springen Sie jeweils mit der Taste 'TAB' zur nächsten Position, um diese dann mit CONTROL-TAB zu löschen. Diesen Vorgang wiederholen Sie, bis die nächste Zeile erreicht ist. Wenn Sie alles richtig gemacht haben, sind alle Tabulatorstellen gelöscht. Wenn nun die Taste 'TAB' betätigt wird, springt der Cursor zum Anfang der nächsten Zeile.

Mit der Tastenkombination SHIFT-TAB können Sie Ihre eigenen Tabulatorpositionen setzen. Probieren Sie dies nun aus. Setzen Sie die Tabulatoren wie folgt:

```
          1         2         3
12345678901234567890123456789012345678
---------x---------x---------x--------
```

Sie müssen dazu den Cursor an die jeweilige Tabulatorstelle bringen und anschließend SHIFT und TAB drücken.

Diese Tabulatoren werden in manchen Programmen, wie z.B. einer Textverarbeitung, benötigt, um Tabellen einzugeben.

Sie haben sicher bemerkt, daß Sie nur 38 anstatt der üblichen 40 Zeichen in einer Zeile eingeben können. Dies liegt daran, daß der linke Rand standardmäßig auf 2 gesetzt ist. Den linken Rand können Sie mit einem Befehl ändern, um 40 Zeichen zu erhalten. Dieser Befehl wird an späterer Stelle beschrieben.

Zum Abschluß die Übersicht der Tabulatorfunktionen:

TAB	**springt zum Tabulator**
CONTROL TAB	**löscht Tabulator**
SHIFT TAB	**setzt Tabulator**

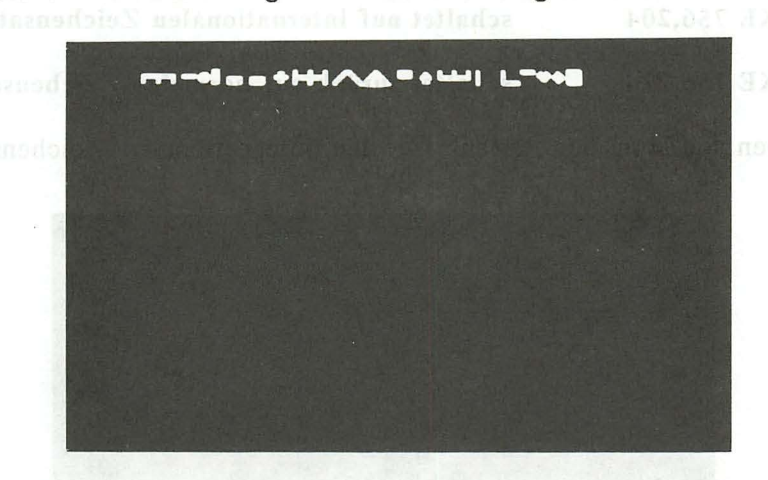
GRAFIKZEICHEN



Ein wesentlicher Unterschied Ihrer ATARI-Tastatur zur Schreibmaschine sind die "versteckten" Sonderzeichen. Diese Zeichen "zaubern" Sie auf den Bildschirm, indem Sie die CONTROL-Taste zusammen mit einer der Grafiktasten betätigen. Folgende Tasten sind Grafiktasten:

Q W E R T Y U I O P
A S D F G H J K L ;
Z X C V B N M , .

Löschen Sie nun den Bildschirm, halten die CONTROL-Taste gedrückt und betätigen nacheinander die oben aufgeführten Tasten. Ihr Bildschirm gleicht dann dem folgenden Bild:



Wenn Sie sehr viele Grafikzeichen eingeben möchten, so bietet die Taste CAPS eine Erleichterung. Wenn Sie die Taste zusammen mit CONTROL betätigen, so steht Ihnen der Grafikzeichensatz zur Verfügung, ohne CONTROL mit der entsprechenden Taste zu drücken. Diesen Zeichensatz schalten Sie mit SHIFT-CAPS wieder zurück.

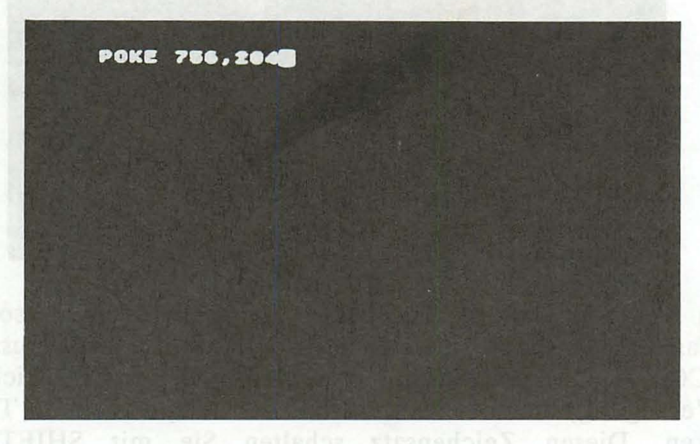
INTERNATIONALE ZEICHEN

Wie bereits zu Anfang erläutert wurde, ist die Tastatur Ihres ATARI der amerikanischen Norm angepaßt. Deutsche Umlaute z.B. suchen Sie deshalb vergebens. Manche Homecomputer werden vor dem Vertrieb in Deutschland "eingedeutscht". Sie erhalten nicht nur deutsche Handbücher, sondern auch eine deutsche Tastatur. Der ATARI hat keine solche Tastatur, sondern internationale Zeichen wie die Grafikzeichen intern gespeichert. Das Ansprechen dieser Zeichen ist jedoch nicht so einfach wie bei den Grafikzeichen. Hierzu ist ein Befehl notwendig, auf den an späterer Stelle näher eingegangen wird. Die Befehle lauten:

POKE 756,204 **schaltet auf internationalen Zeichensatz**

POKE 756,224 **schaltet zurück auf normalen Zeichensatz**

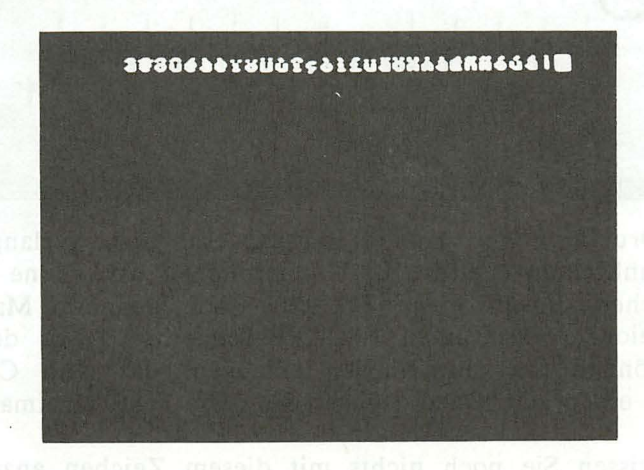
Geben Sie nun den Befehl für den internationalen Zeichensatz ein.



Der Rechner reagiert noch nicht, da Sie diesen Befehl erst mit der Taste 'RETURN' zum Rechner "senden" müssen. Haben Sie dies getan, so stehen Ihnen nun anstatt der Grafikzeichen internationale Zeichen zur Verfügung. Diese Zeichen finden Sie mit gedrückter CONTROL-Taste auf folgenden Tasten:

Q W E R T Y U I O P
A S D F G H J K L ;
Z X C V B N M , .

Halten Sie nun, nachdem Sie den Bildschirm gelöscht haben, die CONTROL-Taste gedrückt und betätigen Sie die zuvor angegebenen Tasten nacheinander. Sie erkennen dann alle verfügbaren, internationalen Zeichen:



Die deutschen Umlaute finden Sie also auf den folgenden Tasten:

ä	**K**
Ä	**;**
ö	**O**
Ö	**L**
ü	**J**
Ü	**P**

Sie erreichen den normalen Zeichensatz entweder mit dem Befehl 'POKE 756,224' oder mit der Taste RESET.

DIE ESC-TASTE



Viele Drucker oder andere Ausgabeeinheiten verlangen für Spezialfunktionen (Fettdruck, Unterstreichen usw.) eine Sequenz von Zeichen, die mit einem ESCAPE-Code beginnen. Man nennt diese Zeichenfolgen auch "ESCAPE-Sequenzen". Mit der Taste 'ESC' können Sie ein Zeichen erzeugen, das dem Code des ESCAPE entspricht. Dazu drücken Sie diese Taste zweimal.

Sicher wissen Sie noch nichts mit diesem Zeichen anzufangen. Doch Sie sollen nicht ewig darüber grübeln, was die Taste oben links am Rechner für eine Aufgabe hat. Was Sie bis jetzt gelernt haben, ist beste Voraussetzung für das spätere Programmieren.

KAPITEL 2:
DER ERSTE BEFEHL

Zur weiteren Erklärung der Tastatur reicht es nun nicht mehr aus, irgendwelche Zeichen auf dem Bildschirm zu schreiben. Die wenigsten unserer Leser erwarben ihren ATARI für diesen Zweck. Für alle diejenigen, die sich im letzten Kapitel etwas langweilten, wird es in diesem Kapitel interessanter. Wir wollen nun den Computer dafür nutzen, wofür er ja einzig und allein vorgesehen ist, nämlich zur Ausführung von Befehlen.

DIE RETURN-TASTE



RETURN ist die wichtigste Taste an Ihrem Rechner. Wenn Sie diese Taste drücken, wird der eingegebene Text an den Rechner übergeben. Der Rechner interpretiert diese Zeile als Befehl und leitet die notwendigen Schritte ein. Nach Abschluß des Befehls meldet sich der Rechner, wie nach dem Einschalten, mit 'READY' wieder. Er ist also wieder bereit zur Aufnahme weiterer Instruktionen. Versuchen Sie nun einmal, den Rechner zu begrüßen. Geben Sie den Text "GUTEN TAG" ein und drücken anschließend die Taste RETURN. Was, meinen Sie, wird Ihr Rechner antworten? Doch sehen Sie selbst.



Hier erscheint die erste Fehlermeldung Ihres ATARI. Die Syntax, also die Zusammensetzung der Zeichen, wird vom Computer nicht verstanden. Zwar gibt es heutzutage Großrechner, die sich weitgehend mit dem Menschen über eingegrenzte Themen "unterhalten" können, jedoch erfordert dies eine Anlage mit hunderttausendfachem Preis und Speicherkapazität. Das, was Sie mit der Tastatur eingeben und mit RETURN "abschicken", wird vom BASIC-Interpreter analysiert und bei fehlerfreier Eingabe ausgeführt.

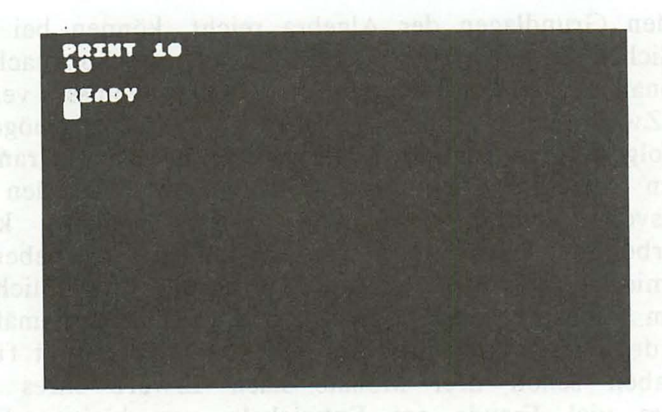
Ihr ATARI ist mit der Programmiersprache BASIC ausgerüstet und versteht somit nur Befehle dieser Sprache. BASIC ist die am meisten verbreitete Programmiersprache, die gerade dem Anfänger optimale Voraussetzungen zum Programmieren bietet. Die Befehle sind weitgehend der englischen Sprache entlehnt. Wie schnell kann ich diese Sprache erlernen? Dies ist eine beliebte Frage, die jedoch nicht hundertprozentig beantwortet werden kann. Viele Kriterien spielen hier eine Rolle. Einmal ist die Lernfähigkeit des Anfängers ausschlaggebend, zum anderen ist die Zeit von Bedeutung, die man am Rechner investiert. Da gibt es Leute, die jede freie Minute dem hochverehrten Computer widmen. Zeitungsanzeigen wie "Verkaufe meinen Computer, Scheidung droht" sind keine Seltenheit mehr. Grundsätzlich gilt es, ein gesundes Maß an Zeit zu opfern, um das Hobby der Programmierung in BASIC zur hellen Freude werden zu lassen. Mit einem gewissen Grundwissen in Mathematik, das vielleicht

bis zu den Grundlagen der Algebra reicht, können bei einem wöchentlichen Zeitaufwand von ca. 10 Stunden schon nach etwa drei Monaten die ersten bombigen Erfolgserlebnisse verbucht werden. Zwar trägt das logische und abstrakte Denkvermögen mit zum Erfolg bei, jedoch wird sich dies auch von Programm zu Programm ständig weiterentwickeln. Zum Erstellen von anspruchsvollen Programmen wie z.B. einer kleinen Textverarbeitung oder einer Dateiverwaltung ist neben der Programmierkenntnis die Programmiererfahrung unerläßlich. Von Programm zu Programm steigert sich erfahrungsgemäß die Qualität des Programmierstils. Ein kleiner Anhaltspunkt für Sie: Viele haben schon drei Monate nach Erwerb ihres ersten Computers viel Freude am Entwickeln von kleinen Spielen gehabt. Solange die Begeisterung da ist, werden auch Sie bald einer der vielen Hobby-Programmierer sein.

DER PRINT-BEFEHL

Ohne diesen Befehl ist kaum ein Programm denkbar. Er übernimmt die gesamte Ausgabe in einem Programm. Ob Sie nun ein Rechenergebnis auf dem Bildschirm oder eine Adresse auf das Kassettenlaufwerk ausgeben wollen, ohne diesen Befehl läuft nichts. Auch die gesamte Druckausgabe wird vom Befehl 'PRINT' übernommen.

Zur Zeit wenden wir uns nur dem Direkt-Modus zu, wir schreiben also noch keine Programme. Direkt-Modus bedeutet, Sie geben Befehle ein, schließen sie mit RETURN ab und erwarten ein sofortiges Ergebnis. Dies wollen wir nun erstmalig ohne Fehlermeldung durchführen. Geben Sie nun den Befehl 'PRINT 10' gefolgt von RETURN ein.



PRINT leitet also alles, was dem Befehl angefügt wird, zum Bildschirm (die Ausgabe auf externe Geräte erfolgt mit einer anderen Form des PRINT). Parameter sind Bestandteile eines Befehls, die genau beschreiben, welche Auswirkungen der Befehl haben soll.

Hier war die Zahl '10' der Ausgabeparameter. 'PRINT 10' bedeutet also 'zeige die Zahl 10 auf dem Bildschirm'. Natürlich können Sie nicht nur Zahlen, sondern auch andere beliebige Zeichen ausgeben. Wie vielseitig dieser Befehl ist, stellen Sie spätestens im weiteren Verlauf dieses Kapitels fest.

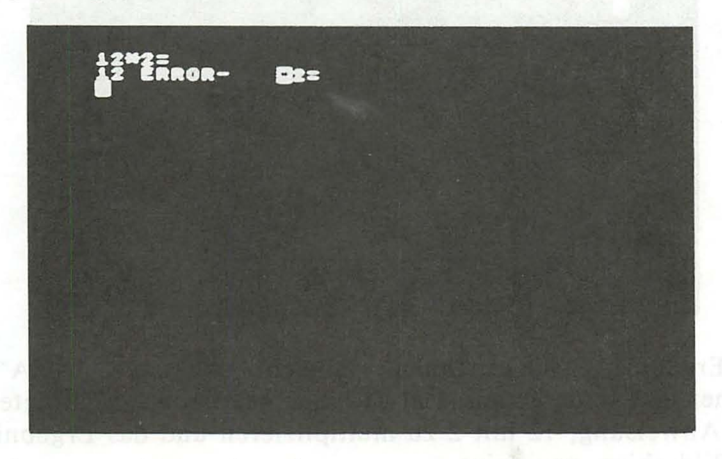
RECHNEN MIT PRINT

Wenn Sie einmal ausrechnen möchten, wieviel Lohnsteuer Sie wohl zurückerstattet bekommen und gerade keinen Taschenrechner zur Hand haben, schalten Sie den ATARI ein! Ich selbst schalte auch oft den Rechner nur für kleine Berechnungen ein. Viele meiner Bekannten haben dann oft ironisch gefragt: "Was, rechnen kann man damit auch?" Das wäre ja auch ein Unding, wenn man mit einem Computer, der immerhin das zwanzigfache eines Taschenrechners kostet, nicht rechnen könnte.

Doch nun zur Sache. Wenden wir uns zunächst den vier Grundrechenarten zu. Hierfür gibt es auf der Tastatur vier Symbole:

+	**für Addition**
-	**für Subtraktion**
*****	**für Multiplikation**
/	**für Division**

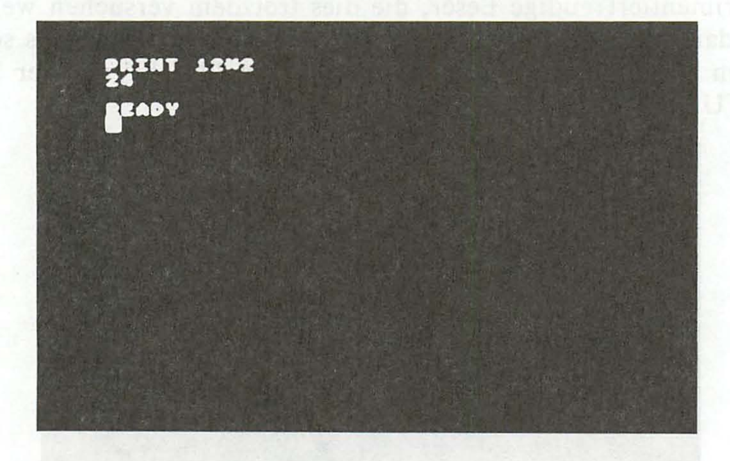
Nur ist das Rechnen mit Ihrem Computer nicht vergleichbar mit einem Taschenrechner. Hier können Sie z.B. nicht '12*2=' eingeben, da bekanntlich ohne Befehl nichts läuft. Es gibt immer experimentierfreudige Leser, die dies trotzdem versuchen werden. Was dann geschieht, ist etwas verwirrend. Versuchen Sie es selbst. Geben Sie '12*2=' ein und schicken diesen "Befehl" mit der Taste 'RETURN' ab.



Was dann geschieht, ist ohne vorzugreifen schwer zu erklären. Der Rechner zeigt nicht das Ergebnis, sondern meldet sich mit einer Fehlermeldung wieder. Der Cursor steht am Anfang der Zeichenkette, die der ATARI nicht ausführen kann. Ich möchte hier nicht zu weit ausholen, doch ich muß etwas vorgreifen. Ein Programm besteht aus mehreren Zeilen, die nach Programmstart hintereinander abgearbeitet werden. Jede dieser Zeilen ist am Anfang mit einer Zahl gekennzeichnet, die die Reihenfolge beim Ablauf bestimmt. In diesem Fall wurde die Programmzeile 12 mit

dem Inhalt '*2=' eingegeben. Dies ist kein korrekter Befehl und deshalb wird diese Fehlermeldung ausgegeben. Versetzen Sie nun Ihren Rechner vor dem weiteren Arbeiten mit der Taste RESET in den Einschaltzustand.

Wie wird nun '12*2' korrekt errechnet und ausgegeben? Die Lösung ist einfach: Geben Sie dazu den Befehl 'PRINT 12*2' ein. Beachten Sie: Immer wenn Sie einen Befehl eingeben, müssen Sie diesen mit RETURN abschließen. Was Sie auf dem Bildschirm schreiben, ist dem Rechner egal. Ihn interessiert nur das, was Sie ihm mit RETURN zur Ausführung überlassen.



Das Ergebnis entlockt Ihnen vielleicht das erste "AHA". Ihr Rechner hat zum ersten Mal für Sie gearbeitet. Er folgte treu Ihrer Anweisung, 12 mit 2 zu multiplizieren und das Ergebnis auf dem Bildschirm anzuzeigen.

Nebenbei möchte ich bemerken, daß Programmiersprachen eigentlich nur kompliziert erscheinen, weil sie die Anweisungen an den Rechner wesentlich verkürzen müssen. Jede Programmiersprache muß bis ins letzte Detail klar organisiert sein. Eine Programmiersprache, die Anweisungen wie 'RECHNE 2*12 UND ZEIGE ERGEBNIS' akzeptiert, gibt es nicht und wird es auch nie geben. Verdrängen Sie bitte derartige Erwartungen. Ganz so einfach ist das Programmieren auch nicht. Wenn Sie es beherrschen, dürfen Sie sich nicht umsonst selbst auf die Schulter klopfen.

Entgegen dem BASIC mancher anderer Computer braucht das Leerzeichen hinter dem PRINT hier nicht angegeben werden. Doch nicht nur hier, sondern grundsätzlich nach jedem Befehl können Leerzeichen ignoriert werden. Dies werden Sie im weiteren Verlauf des Buches schnell erlernen. Doch nun weiter mit den Grundrechenarten. Berechnen Sie hintereinander '12+2', '12-2', '12*2' und '12/2'. Ihr Bildschirm sollte nachher der folgenden Abbildung gleichen:

```
PRINT 12+2
14

READY
PRINT 12-2
10

READY
PRINT 12*2
24

READY
PRINT 12/2
6

READY
```

Haben Sie es geschafft? Gut, dann dürfen Sie wieder eine Sprosse höher auf die Erfolgsleiter zum Hobby-Programmierer steigen. Nein? Dann lesen Sie bitte den letzten Abschnitt nochmals aufmerksam durch.

Neben solch simplen Berechnungen können Sie auch lange Berechnungen von bis zu 255 Zeichen anstellen! Beachten Sie jedoch die Hierarchie bei solchen Berechnungen: Punktrechnung geht stets vor Strichrechnug. Geben Sie nun dem Rechner die Aufgabe, das Ergebnis von '1+2+3+4+5+6+7+8+9' zu ermitteln.



DIE KLAMMERRECHNUNG

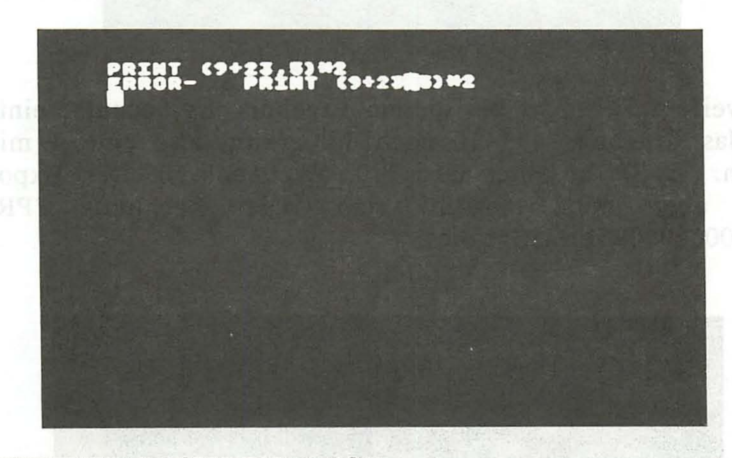


Der ATARI bietet die Möglichkeit, beliebige Kettenrechnungen durchzuführen. Ein praktisches Beispiel: Es soll der Gesamtpreis von drei Teppichstücken ermittelt werden. Der Quadratmeterpreis beträgt 23.80 DM. Das erste Stück umfaßt 2.45 m * 2.80 m, das zweite Stück 4.50 m * 3.85 m und das dritte Stück 2.75 m * 4.80 m. Zusätzlich soll 14% Mehrwertsteuer addiert werden. Wie hoch ist nun der Gesamtpreis für diese drei Teppichzuschnitte? Wenn Sie die Klammern richtig setzen, genügt ein PRINT-Befehl:

```
PRINT (2.45*2.8+4.5*3.85+2.75*4.8)*23.8*1.14
```

Das sieht komplizierter aus, als es eigentlich ist. Innerhalb der Klammer werden die Quadratmeter ausgerechnet. Dann wird die Quadratmeteranzahl mit dem Quadratmeterpreis multipliziert. Schließlich wird noch die Mehrwertsteuer hinzugerechnet. Und das alles mit einem Befehl! Bei korrekter Eingabe erhalten Sie den Wert 1014.32982, also den Preis 1014.33 DM.

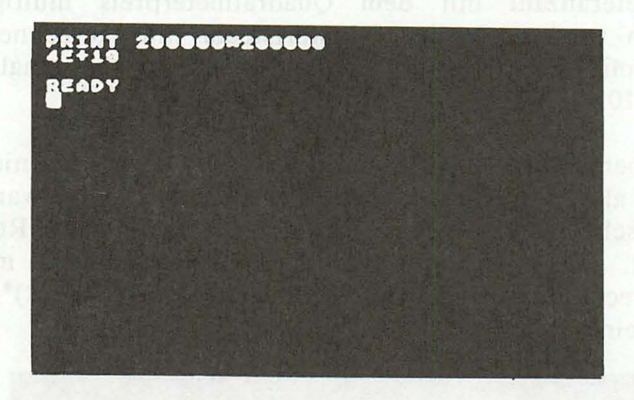
Wie Sie bereits erkannt haben, wird der Punkt und nicht das Komma als Dezimalpunkt benutzt. Das ist zwar eine amerikanische Norm, wird jedoch auch an deutschen Rechnern praktiziert. Wenn Sie das Komma in Verbindung mit der Klammerrechnung eingeben, z.B. "PRINT (9+23,8)*2", so erscheint eine Fehlermeldung:



EXPONENTIALSCHREIBWEISE

Der Rechner stellt das letzte Ergebnis mit 5 Nachkommastellen dar. Es ist eine sogenannte Fließ- oder Gleitkommazahl. Grundsätzlich wird auf neun Stellen genau gerechnet. Ergibt sich eine Zahl mit drei Stellen vor dem Komma, so wird automatisch auf 6 Stellen nach dem Komma gerechnet. Sollte die Zahl 999999999 überschreiten, so wird wieder auf neun Stellen genau gerechnet. Eine größere Zahl erhält dann einen Zusatz, der die

Größe des echten Ergebnisses kennzeichnet. Geben Sie z.B. einmal die Rechnung 'PRINT 200000*200000' ein. Das Ergebnis ist eine elfstellige Zahl, die nicht mehr normal dargestellt werden kann.



Verzweifeln Sie nicht bei diesem Ergebnis. Es bedeutet einfach, daß das Ergebnis '4 * 10 hoch 10' ergibt, also eine 4 mit 10 Nullen. 'E+10' bedeutet Basis 10, Exponent 10. Der Exponent kann aber auch negativ sein. Die Rechnung 'PRINT 9/30000000000' bestätigt dies.

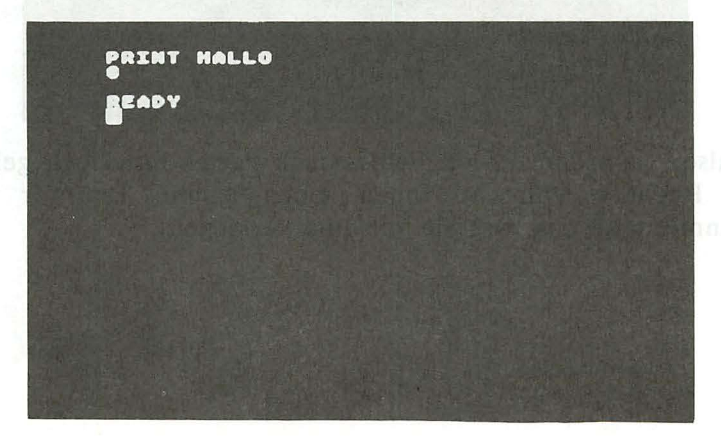


Obwohl das Ergebnis eigentlich 0.0000000003 lauten sollte, benutzt der Rechner die Exponentialschreibweise. '3E-10' ist gleichbedeutend mit '3 * 10 hoch -10'. Die Zahl '3' befindet sich also an zehnter Stelle nach dem Komma. Das ist vorerst alles, was Sie über das Rechnen mit den vier Grundrechenarten im Direkt-Modus wissen sollten. Weitere mathematische Funktionen finden Sie in Ihrem Handbuch zum ATARI.

TEXTAUSGABE MIT PRINT



Neben Zahlen können auch Texte, sogenannte "Strings" (Zeichenketten), mit PRINT ausgegeben werden. Voreilige Versuche wie 'PRINT HALLO' werden kläglich scheitern.



Obwohl der Befehl nicht den gewünschten Erfolg verbuchen kann, erscheint keine Fehlermeldung. Der Rechner hat den Befehl also ausgeführt. Nur was hat er ausgeführt? Woher die Null? Dies sind Probleme, die bei den ersten Gehversuchen mit BASIC immer wieder auftreten. Die Fragen sind, ohne vorzugreifen, schwer zu beantworten. Kurz erklärt, wird der Inhalt einer Variablen (rechnerinterner Speicher) angezeigt. Die Bezeichnung dieses Speichers ist 'HALLO'. Da wir in der Variablen nichts angelegt haben, wird der Wert 0 ausgegeben. Sollten Sie dies nicht ganz verstanden haben, so ist das kein Grund zur Beunruhigung. Die Variablen werden später noch ausführlich behandelt.

Doch wie kann man nun Zeichenketten (bleiben wir im weiteren Verlauf beim Fachwort "Strings"), wie kann man Strings mit PRINT ausgeben? Die Lösung ist einfach: Strings werden in Anführungszeichen eingeschlossen. Setzen Sie dies nun in die Tat um und ändern den letzten Befehl. Geben Sie den String 'HALLO' wie folgt ein:

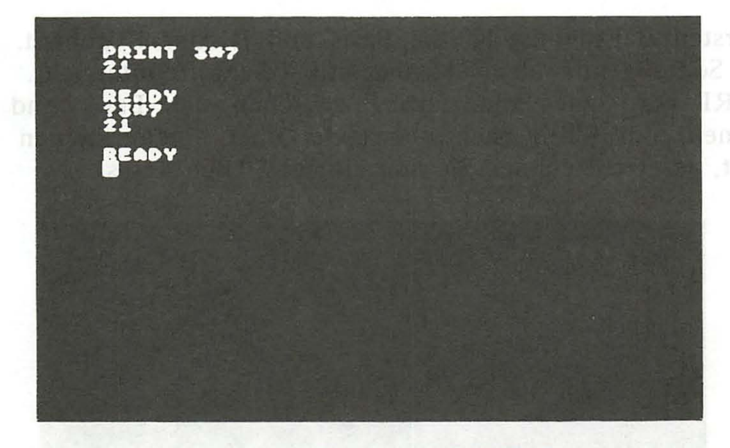


Na also, nach dem ersten Fehlversuch hat es nun doch geklappt. Der Rechner wünscht Ihnen einen guten Tag (er macht bekanntlich all das, was Sie von Ihm verlangen).

VEREINFACHTE PRINT-EINGABE



Geben Sie anstelle des Befehls PRINT ein Fragezeichen ein, so ersparen Sie sich damit etwas Arbeit, zumal der Befehl PRINT der am häufigsten verwendete Befehl ist. Es ist also egal, ob Sie nun 'PRINT 3*7' oder '? 3*7' eingeben. Probieren Sie es selbst einmal aus. Hinter dem Fragezeichen ist kein Leerzeichen erforderlich



Es liegt nun an Ihnen, ob Sie den PRINT-Befehl ausschreiben oder das Fragezeichen verwenden. Bei Berechnungen, die "mal eben" eingetippt werden, ist das Fragezeichen recht praktisch. Man kann damit ebenso schnell rechnen wie mit einem Taschenrechner.

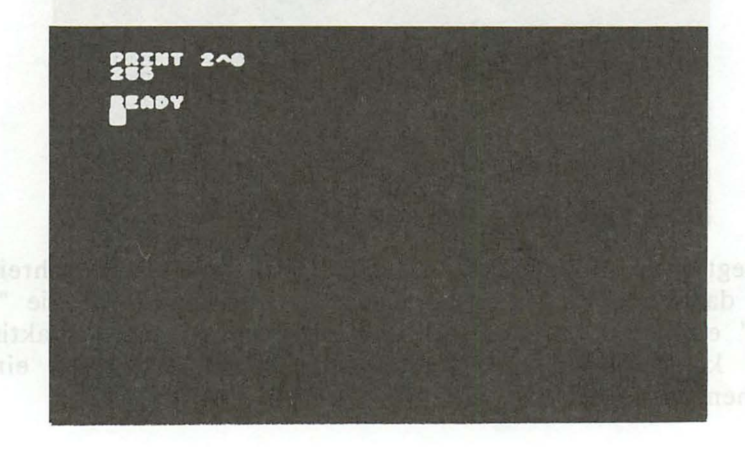
POTENZIERUNG



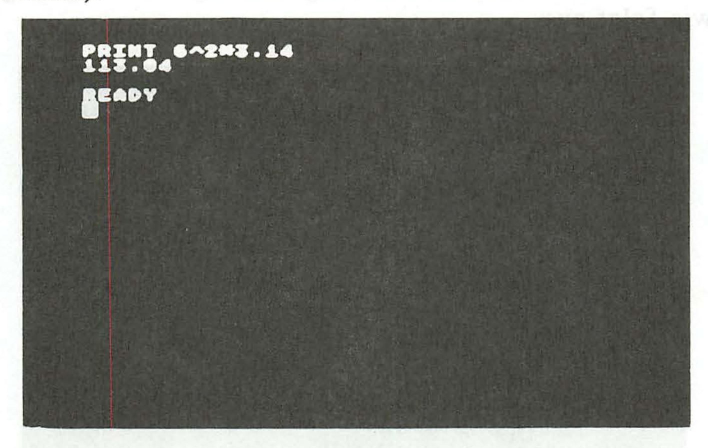
Potenzierung bedeutet, eine Zahl mit sich selbst malzunehmen. Der Exponent (die Hochzahl) bestimmt, wie oft die Basis (Grundzahl) mit sich selbst malgenommen werden soll.

2 hoch 3 bedeutet 2*2*2, also 8
10 hoch 4 bedeutet 10*10*10*10, also 10000

Im ersten Beispiel ist '2' die Basis und '3' der Exponent. Doch nun Schluß mit dem Mathematik-Nachhilfeunterricht. Beim ATARI wird zur Potenzierung zwischen der Basis und dem Exponent ein 'Pfeil nach oben', der eher einem kleinen Dach ähnelt, gesetzt. Rechnen Sie nun einmal 2 hoch 8 aus.



Das war sicher nicht schwer. Berechnen Sie nun die Kreisfläche eines Kreises mit dem Durchmesser von 12 cm. Die Formel lautet: Kreisfläche = Radius hoch 2 mal PI (Radius ist der halbe Durchmesser).

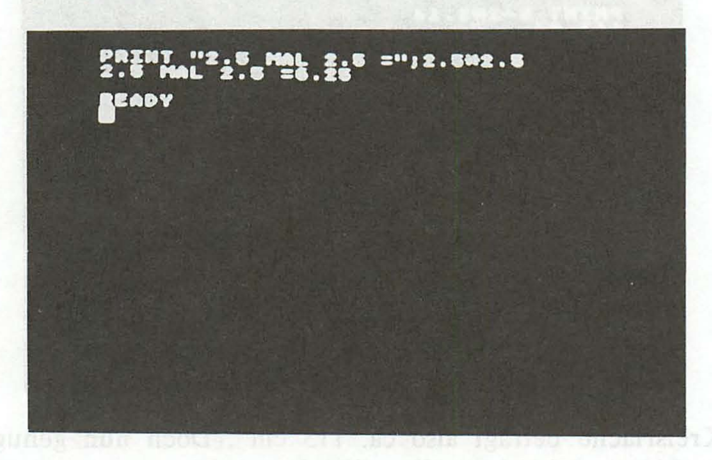


Die Kreisfläche beträgt also ca. 113 cm^2. Doch nun genug von der trockenen Mathematik. Wenden wir uns nun wieder interessanteren Sachen zu.

KOMBINIEREN VON STRINGS MIT ZAHLEN



Die Tasten ';' und ',' spielen eine große Rolle, wenn Strings und Zahlen kombiniert werden sollen. Aber welche? Wenn Sie z.B. eine Zeile wie "2.5 mal 2.5 = 6.25" ausgeben wollen, bei der das Ergebnis gleichzeitig errechnet werden soll, so wird ein String und eine anschließende Berechnung benötigt. Die Lösung sieht dann wie folgt aus:

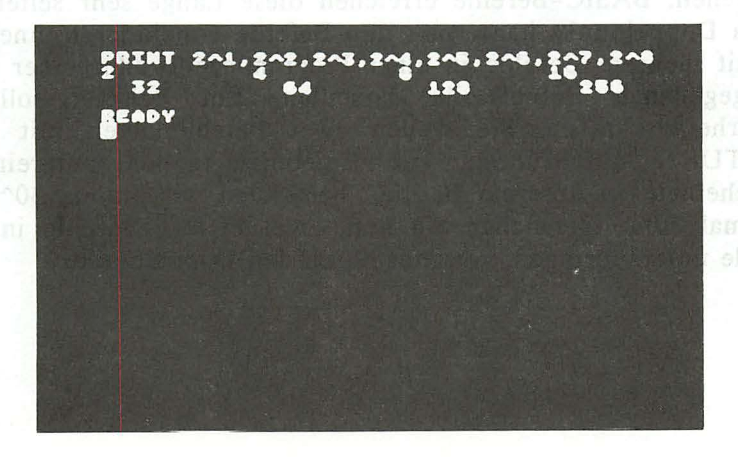


Das Semikolon trennt also Strings von Zahlen. Doch nicht nur das, auch Strings und Zahlen bzw. Berechnungen können untereinander durch das Semikolon getrennt werden. So können Sie z.B. mehrere Berechnungen mit einem PRINT-Befehl ausführen und die Ergebnissse nebeneinander anzeigen. Versuchen Sie nun, die Potenzen der Zahl '2' vom Exponent '1' bis zum Exponent '8' zu berechnen und nebeneinander in einer Zeile anzuzeigen.



Ein mit SEMIKOLON abgeschlossener PRINT-Befehl hat zur Folge, daß ein weiterer PRINT-Befehl nicht in der nächsten Zeile, sondern in der gleichen Zeile hinter dem ersten PRINT erfolgt. Doch hierzu erfahren Sie später mehr.

Da die mit Semikolon getrennten Zahlen nicht auseinanderzuhalten sind, geben Sie nochmals den letzten Befehl ein, ersetzen aber alle Semikoli durch Kommata.



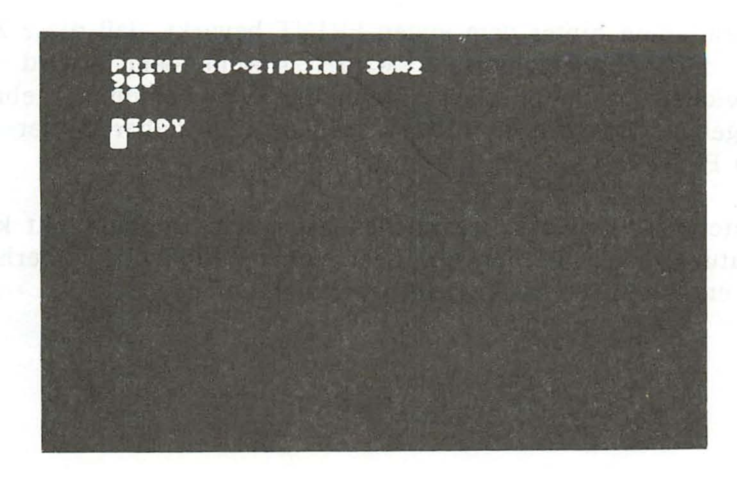
Das Komma trennt auch die einzelnen Berechnungen, aber erzeugt einen gößeren Abstand. Wenn Sie die Abstände auszählen, so werden Sie feststellen, daß sie zwischen den Ziffern jeweils 10

Zeichen betragen. Dadurch wird eine leicht formatierte Ausgabe von Zahlenkolonnen ermöglicht. Zur Trennung von Strings wird das Komma nur selten genutzt.

TRENNEN VON BEFEHLEN



Eine Befehlszeile kann aus maximal 114 Zeichen (3 Zeilen) bestehen. BASIC-Befehle erreichen diese Länge sehr selten. Mit dem Doppelpunkt kann man nun Befehle voneinder trennen und somit mehrere Befehle in einer Programmzeile oder einer direkt eingegebenen Befehlszeile darstellen. Ein Beispiel soll hier Klarheit schaffen. Sie wollen zwei Berechnungen mit einem RETURN durchführen. Die Ergebnisse sollen untereinander erscheinen. In unserem Beispiel berechnen wir einmal 30^2 und einmal 30*2. Versuchen Sie nun, zwei PRINT-Befehle in einer Zeile unterzubringen, getrennt durch den Doppelpunkt.



Sie können natürlich auch drei oder vier Befehle eingeben und durch Doppelpunkt trennen. Es ist nur darauf zu achten, daß 114 Zeichen nicht überschritten werden. Kurz vor Ende der dritten Befehlszeile meldet der Rechner mit einem akustischen Signal, daß das Maximum nach weiteren 8 Zeichen erreicht ist.

Hier läßt sich auch sehr gut die Wirkung des Semikolons nach dem ersten PRINT-Befehl verdeutlichen. Löschen Sie den Bildschirm und geben Sie die beiden PRINT-Befehle nochmals ein. Setzen Sie hinter das erste PRINT ein Semikolon.

```
PRINT 30^2;" ";:PRINT 30*2
900 60
READY
```

Das Semikolon hinter dem ersten PRINT bewirkt, daß diese Zeile noch nicht abgeschlossen wird. Anschließend wird ein Leerzeichen zur optischen Trennung der beiden Ergebnisse ausgegeben. Das zweite PRINT erfolgt unmittelbar hinter dem ersten PRINT.

Beachten Sie, daß das Leerzeichen nach dem Doppelpunkt keine Bedeutung hat. Ob hier ein Leerzeichen, fünf oder überhaupt keins enthalten ist, hat keine Auswirkung auf den Befehl.

KAPITEL 3:
DAS ERSTE PROGRAMM

Mit dem ersten Programm ist nicht etwa die ARD gemeint, sondern Ihre ersten Gehversuche bei der Programmierung in BASIC. Verdrängen Sie Ihren eventuellen Unmut, dieses "heiße Eisen" anzufassen. BASIC ist eine Programmiersprache für den Einsteiger, vor der niemand allzu großen Respekt zu haben braucht. Die Tatsache, daß in Kaufhäusern und Computerläden oft Kinder anzutreffen sind, die nach der Schule ihre Zeit mit BASIC vertreiben, sollte die letzten Hemmungen beiseite schieben. Seien Sie kein ewiger Anwender, der seine ganze Zeit den Fertigprogrammen widmet. Mal ehrlich, haben Sie nicht auch oft daran gedacht, eigene Programme zu entwickeln? Dieses und das folgende Kapitel soll für Sie ein kleiner Schritt in die große und interessante Welt der Programmierung sein.

Hinweis: Der ATARI 130XE besitzt zwar 128 KByte Hauptspeicher, jedoch sind 64 KByte davon nicht ohne weiteres verfügbar. Diese sogenannte zweite "Bank" ist nur mit wirklichem Aufwand unter Einsatz etlicher PEEK's und POKE's unter BASIC zu nutzen - also etwas für "Freaks".

EIN PROGRAMM, WAS IST DAS?

Auf Fachchinesisch wird ein Programm als eine "Folge von Befehlen zur Lösung einer bestimmten Aufgabe" definiert. Diese Folge von Befehlen ist es also, die in logisch richtiger Zusammensetzung ein Programm ergibt. Der Weg zu einem Programm führt also von der Aufgabenstellung über die logische Befehlsfolge zum Ziel, nämlich dem Programm. Logisch richtig bedeutet, daß nicht nur sämtliche BASIC-Befehle bekannt sein müssen, sondern daß sie erst durch ausgetüftelte Zusammenstellung zur Problemlösung führen. So ist es z.B. sinnlos, eine Unmenge von englischen Vokabeln zu kennen, wenn

Sie diese nicht durch sinnvolle Zusammensetzung zur Kommunikation nutzen können.

DIE ZEILENNUMMERIERUNG

Ein Programm ist also eine Folge von Befehlen. Doch wodurch wird die Reihenfolge bestimmt? Nun, bei der Programmiersprache BASIC wird jede Befehlszeile (auch Statement genannt) mit einer Nummer versehen, die die Reihenfolge bestimmt, mit der das Programm ablaufen wird. Stellen Sie sich z.B. Ihr im Gehirn gespeichertes Programm zur Lösung der Rechenaufgabe '48/12' mit dem Taschenrechner vor. Sie werden feststellen, daß auch hier eine gewisse Reihenfolge eingehalten werden muß:

1. Suche Taschenrechner.
2. Schalte Rechner ein.
3. Wenn Batterie leer, dann Punkt 12.
4. Drücke die Taste '4'.
5. Drücke die Taste '8'.
6. Drücke das Operationzeichen für Divison.
7. Drücke die Taste '1'.
8. Drücke die Taste '2'.
9. Drücke die Taste '='.
10. Lese Ergebnis ab.
11. Speichere Ergebnis im Kleinhirn.
12. Schalte Rechner aus.

Erstaunlich, in wieviel Einzelschritten doch eine so simple Tätigkeit zerlegt werden kann. Sie erkennen, daß alle Schritte numeriert sind, um die Reihenfolge festzulegen. Doch dies ist nicht der einzige Grund. Der Schritt Nummer 3 schneidet eine wichtige Programmierlogik an, den Sprungbefehl. Wenn eine bestimmte Bedingung erfüllt ist (Batterie leer?), verzweigt das Programm zum Schritt 12. Diese Zahl ist die sogenannte Adresse des Befehls. Ohne Zeilennummer wäre es also nicht möglich, bestimmte Befehle gezielt anzuspringen.

Wir haben im vorherigen Kapitel bereits einen Befehl kennengelernt, den PRINT-Befehl, mit dem Daten auf dem Bildschirm ausgegeben werden. Diesen Befehl setzten wir jedoch nur im Direkt-Modus ein, d.h. der Befehl wird nach dem RETURN direkt ausgeführt. Wir wollen nun eine Folge von drei PRINT-Befehlen als Programm codieren. Was ist zu tun? Richtig, jeder Befehl erhält eine Zeilennummer, die bestimmt, in welcher Reihenfolge das Programm abgelaufen wird.

DIE ZEILENNUMMERN DÜRFEN BEIM ATARI IM BEREICH VON 0 BIS 32767 LIEGEN. DIE SCHRITTWEITE IST OHNE BEDEUTUNG.

Die Schrittweite bestimmt, um welche Zahl die Folgezeile größer als die vorherige Zeile ist. So kann ein vierzeiliges Programm z.B. aus den Zeilennummern 1, 8, 10, 20 bestehen. Auch die Reihenfolge 100, 200, 300, 400 ist möglich. Die Schrittweite darf also innerhalb eines Programms variieren.

Doch wozu ist eine Schrittweite von größer als 1 nützlich. Ist doch klar, wenn zwischen zwei Zeilen in einem Programmn noch eine Zeile eingefügt werden soll, so muß die Schrittweite zwischen diesen beiden Zeilen größer als 1 sein, da Zeilennummern immer ganzzahlig sind. Zwischen der Zeile 11 und der Zeile 12 kann keine Zeile mehr eingefügt werden. Zwischen den Zeilennummern 11 und 15 jedoch gibt es mehrere Möglichkeiten, eine Zeile "einzumogeln". Diese Zeile erhält die Nummer 12, 13 oder 14 und wird automatisch zwischen den Zeilen 11 und 15 eingeordnet. In der Praxis hat sich eine Schrittweite von 10 bewährt.

Doch nun zu unserer Aufgabe, die wie folgt aussehen soll:

1. Lösche den Bildschirm.
2. Gebe den Text "18 GETEILT DURCH 6 ERGIBT:" aus.
3. Gebe das Ergebnis unmittelbar hinter dem Text aus.

Zur Lösung der Aufgabe sollen drei BASIC-Zeilen verwendet werden. Wie geht man nun diese Aufgabenstellung an? Zunächst muß die erste Zeilennummer ausgewählt werden. In unserem

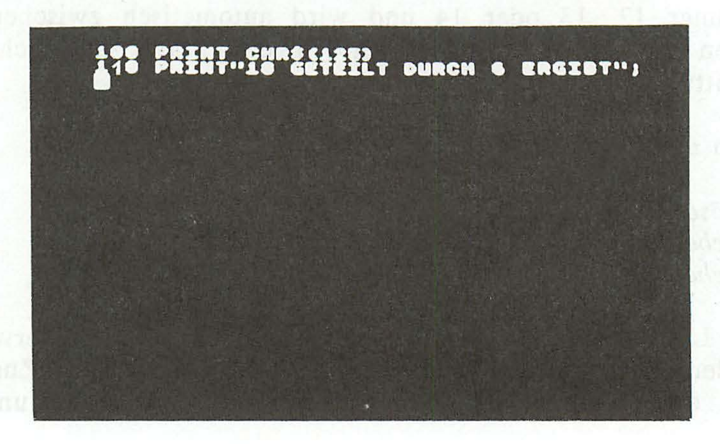
Beispiel soll dies die Zeile 100 sein. Wir geben dazu die Zahl 100 gefolgt von dem Befehl zum Löschen des Bildschirms ein:

'PRINT CHR$(125)' LÖSCHT DEN BILDSCHIRM

Die Zahl '125' ist der Code zum Löschen des Bildschirms. Mit der Funktion 'CHR$' kann jedes beliebige Zeichen mit seinem Code angesprochen werden. Mehr zu dieser Funktion an späterer Stelle.

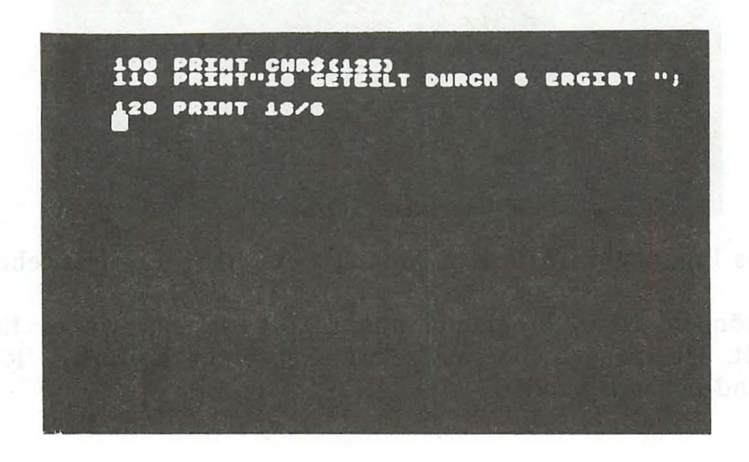


Dies ist nun die erste Zeile unseres Programms. Die zweite Zeile soll eine Zeichenkette ausgeben. Dies erfolgt bekanntlich mit dem Befehl PRINT. Wir arbeiten mit einer Schrittweite von 10. Die zweite Zeile erhält demnach die Nummer 110. Geben Sie diese Zeile nun ein.



Hinter dem String wird hier ein Semikolon gesetzt, damit der folgende PRINT noch in dieser Zeile, also hinter dem Text ausgegeben wird.

Die dritte Zeile sollten Sie nun alleine ermitteln. Es soll 18/6 berechnet und ausgegeben werden.

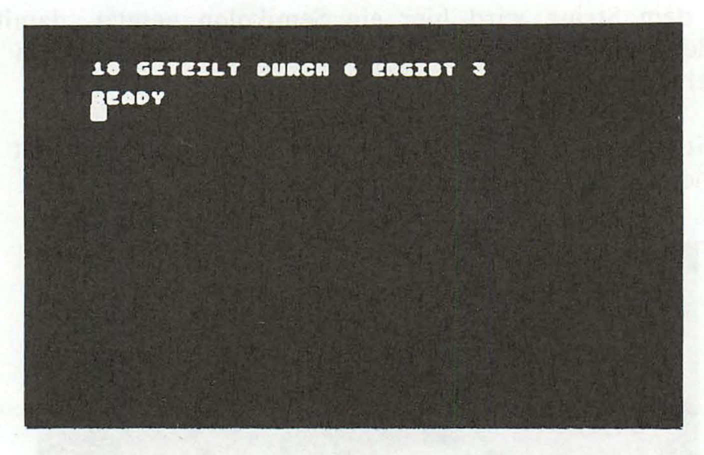


PROGRAMMSTART

Der ATARI hat die soeben eingegebenen Programmzeilen nicht nur auf dem Bildschirm, sondern auch in seinem BASIC-Speicher festgehalten. Auch wenn Sie den Bildschirm löschen, geht das Programm nicht verloren. Es kann zu jeder Zeit gestartet werden.

DER BEFEHL 'RUN' STARTET EIN BASIC-PROGRAMM

Geben Sie nun den Befehl 'RUN' ein und beachten Sie, was auf dem Bildschirm geschieht.



Dieses Programm ist für den Anfänger ein erstes Erfolgserlebnis.

Sie können dieses Programm nun so oft starten, wie es Ihnen gefällt. Überzeugen Sie sich selbst: Geben Sie nochmals 'RUN' ein und nochmals und....

PROGRAMMÄNDERUNG

Es kann durchaus vorkommen, daß an einem Programm Änderungen vorgenommen werden müssen. Stellen Sie sich vor, Sie möchten mit dem zuletzt eingegebenen Programm nicht 18/6, sondern 24/6 berechnen. Dazu müssen Sie nicht etwa alle drei Zeilen nochmals eingeben, sondern Sie können die bestehenden Zeilen an den entsprechenden Stellen abändern. Dazu müssen Sie aber erst wieder das Programm auf den Bildschirm holen.

DER BEFEHL 'LIST' ZEIGT DIE PROGRAMMZEILEN AUF DEM BILDSCHIRM AN.

"Schon wieder ein neuer Befehl" werden Sie vielleicht denken. Doch dies wird mit Sicherheit nicht der letzte sein. Um in BASIC zu programmieren, müssen Sie mit den wichtigsten Befehlen vertraut sein.

Geben Sie nun den Befehl 'LIST' gefolgt von der Taste RETURN ein und beobachten, was geschieht. Ihre soeben eingegebenen Programmzeilen erscheinen auf dem Bildschirm.

Wollen Sie nur Teile des Programms auflisten, so müssen Sie dem Befehl LIST die Anfangs- und Endzeile anhängen. Dazu die folgenden Beispiele:

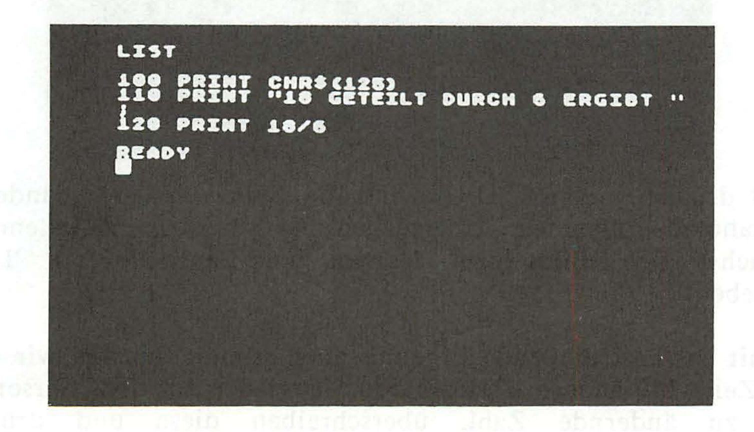
LIST 10,100 *listet die Zeilen 10 bis 100*
LIST 100,899 *listet die Zeilen 100 bis 899*

Was Sie noch wissen sollten: Wenn größere Programme aufgelistet werden, so "scrollt" der Bildschirm. Scrollen bedeutet, es werden von unten Zeilen nachgeschoben, wobei gleichzeitig die obersten Zeilen verschwinden. Um dieses Scrollen anzuhalten, drücken Sie 'CONTROL' zusammen mit der Taste '1'. Ein nochmaliges Drücken dieser Kombination setzt die Ausgabe fort.

CONTROL '1' HÄLT DIE AUSGABE AN UND STARTET SIE WIEDER.

DIE TASTE 'BREAK' BRICHT DIE AUSGABE AB.

Kommen wir nun auf die Programmänderung zurück. Um die Zeilen nun zur Berechnung von 24/6 abzuändern, müssen Sie diese zunächst auf dem Bildschirm auflisten.



Nun bewegen Sie den Cursor auf das erste zu ändernde Zeichen in der Zeile 110.

```
LIST

100 PRINT CHR$(125)
110 PRINT "18 GETEILT DURCH 6 ERGIBT "
120 PRINT 18/6

READY
```

Jetzt überschreiben Sie die '18' mit der '24'. Dazu drücken Sie einfach die Tasten '2' und '4'.

```
LIST

100 PRINT CHR$(125)
110 PRINT "24 GETEILT DURCH 6 ERGIBT "
120 PRINT 18/6

READY
```

Jetzt drücken Sie RETURN, und die Zeile wird im geänderten Zustand übernommen. Überzeugen Sie sich davon, indem Sie zunächst den Bildschirm löschen und anschließend 'LIST' eingeben.

Damit das anschließende Ergebnis auch stimmt, müssen wir noch die Zeile 120 ändern. Dazu fahren Sie wieder mit dem Cursor auf die zu ändernde Zahl, überschreiben diese und drücken

anschließend RETURN. Starten Sie nun nochmals das Programm mit 'RUN'.

Die Änderung der eingegebenen Programmzeilen ist beim ATARI recht einfach. Wenn Sie die Cursortasten beherrschen, wird Ihnen eine Programmänderung keine Schwierigkeiten bereiten.

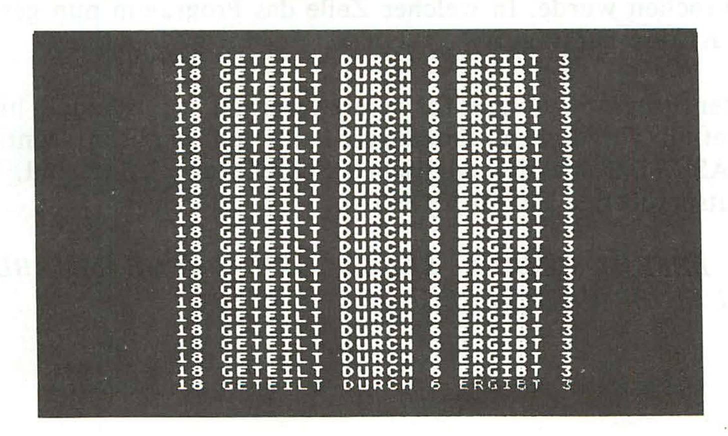
VERZWEIGUNG

Programme werden nur selten wie im letzten Beispiel stur von vorne nach hinten abgearbeitet. Oft werden, meist zu bestimmten Bedingungen, Zeilen übersprungen und an anderer Stelle fortgesetzt. Hier ist ein Sprung erforderlich.

DER BEFEHL 'GOTO' SPRINGT ZU EINER ANGEGEBENEN ZEILE.

In BASIC gibt es den Befehl 'GOTO', der eine beliebige Zeilennummer anspringt.

Lassen Sie uns dies gleich in die Praxis umsetzen. Bekanntlich endet das Programm nach der Zeile 120. Was geschieht, wenn wir danach, z.B. in Zeile 130 wieder in die Zeile 110 springen? Ich behaupte, es wird ununterbrochen "18 GETEILT DURCH 6 ERGIBT 3" ausgegeben. Doch sehen Sie selbst. Geben Sie die Zeile '130 GOTO 110' ein und starten das Programm mit 'RUN'.

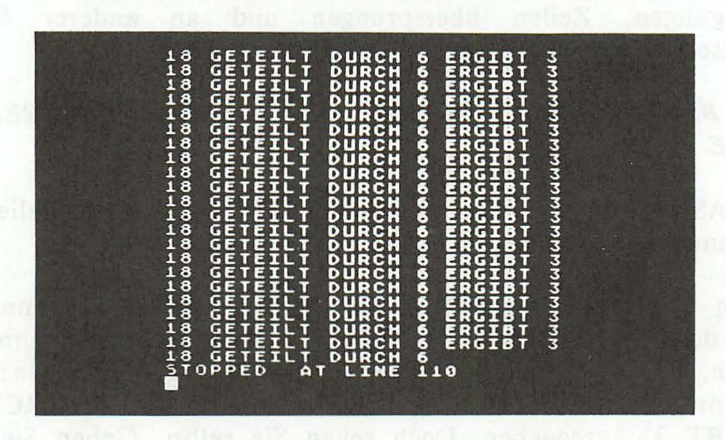


Wir haben mit der Zeile 130 eine sogenannte Endlosschleife erstellt, da das Programm ständig die Zeilen 110 bis 130 durchläuft. Wenn wir nun nicht eingreifen, läuft das Programm noch stundenlang so weiter.

DIE KOMBINATION 'CONTROL' '1' UNTERBRICHT DIE AUSGABE UND SETZT SIE BEI ERNEUTER BETÄTIGUNG WIEDER FORT.

DIE TASTE 'BREAK' BRICHT DEN PROGRAMMABLAUF AB.

Drücken Sie nun die Taste 'BREAK', so hält das Programm an.



Der Rechner meldet nun, in welcher Zeile das Programm unterbrochen wurde. In welcher Zeile das Programm nun gestoppt wird, ist rein zufällig.

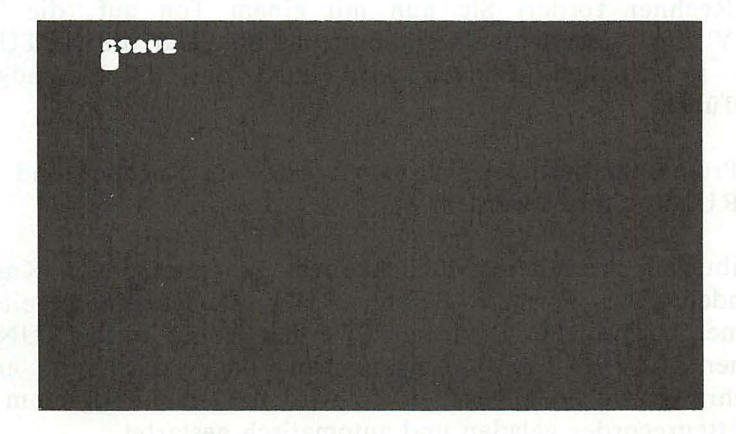
Soll der Programmablauf fortgesetzt werden, so ist auch hierfür ein Befehl vorgesehen. Fortsetzen heißt auf englisch "continue". Da BASIC-Befehle der englischen Sprache angelehnt sind, heißt der entsprechende Befehl 'CONT'.

DER BEFEHL 'CONT' SETZT DEN PROGRAMMABLAUF NACH EINER UNTERBRECHUNG FORT.

Geben Sie nun diesen Befehl ein. Das Programm befindet sich wieder in der Endloschleife und kann wiederum mit der Taste 'BREAK' abgebrochen werden.

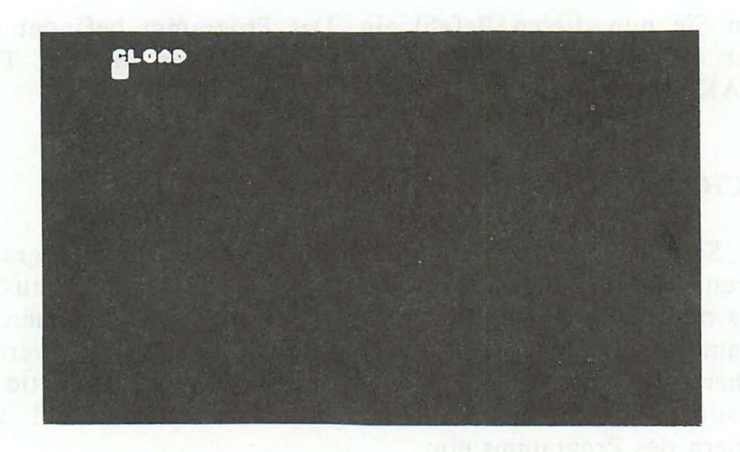
SPEICHERN UND LADEN VON PROGRAMMEN

Wenn Sie den Rechner nun ausschalten, so ist das Programm verloren. Da Sie sicherlich nicht nach jedem Einschalten des Geräts das gewünschte Programm eintippen möchten, können die Programme z.B. auf dem Kassettenrecorder abgelegt werden. Speichern wir nun unser Programm. Legen Sie eine Kassette ein und spulen diese zurück. Nun geben Sie den Befehl zum Speichern des Programms ein:



Der Rechner gibt zwei Töne von sich und signalisiert damit, daß Sie die Tasten PLAY und RECORD des Recorders drücken sollen. Wenn Sie dies getan haben, drücken Sie die RETURN-Taste des Rechners, und das Programm wird gespeichert. Der ATARI meldet sich anschließend mit READY wieder.

Wenn Sie das Programm nun wieder laden und starten möchten, so spulen Sie die Kassette wieder en. Wenn das gewünschte Programm das erste Programm auf der Kasette ist, so spulen Sie einfach die Kassette zurück und geben den Befehl 'CLOAD' ein. Dieser Befehl lädt das erste Programm auf der Kassette.



Der Rechner fordert Sie nun mit einem Ton auf, die Taste 'PLAY' am Kassettenlaufwerk und anschließend die RETURN-Taste zu drücken. Danach wird dann der Befehl endgültig ausgeführt.

Das Programm befindet sich nun wieder im Rechner und kann mit 'RUN' gestartet werden.

Es gibt jedoch weitere Möglichkeiten, Programme von Kassette zu laden. Der normale Befehl 'RUN' startet ein bereits im Rechner befindliches Programm. Werden jedoch nach 'RUN' die Zeichen 'C:' in Anführungszeichen oder nur mit erstem Anführungszeichen angegeben, so wird das erste Programm vom Kassettenrecorder geladen und automatisch gestartet.

Die folgenden Beispiele sollen Ihnen die verschiedenen Möglichkeiten zum Laden und Speichern eines Programs verdeutlichen:

Befehl	**Funktion**
RUN"C:"	lädt das erste Programm und startet es anschließend.
RUN"C:TEST 1"	lädt das Programm mit dem Namen 'TEST1' und startet es anschließend.
RUN	startet ein bereits im Speicher befindliches Programm.
CLOAD oder LOAD "C:"	lädt das erste Programm in den Speicher.
LOAD"C:TEST 1"	lädt das Programm mit dem Namen 'TEST1' in den Speicher.
CSAVE oder SAVE"C	speichert das Programm ohne Namen.
SAVE"C:TEST 1"	speicher das Programm unter dem Namen "TEST 1".

Einige Befehle sehen Sie hier zum ersten Mal. So kann den Befehlen 'LOAD', 'SAVE' und 'RUN' ein Programmname zugefügt werden. Es wird dann auf der Kassette dieses bestimmte Programm gesucht und erst nach dem Auffinden geladen. Damit haben Sie die Möglichkeit, mehrere Programme auf einer Kassette abzulegen und gezielt zu laden. Es ist jedoch sinnvoll, dann den jeweiligen Stand des Bandzählwerkes zu notieren. Sie können dann vor dem Laden an die entsprechende Stelle vorspulen.

Die zweite Neuigkeit für Sie ist, daß mit dem Befehl 'RUN' eine Zeilennummer angegeben werden kann. Das Programm wird dann nicht wie gewohnt bei der ersten Zeile, sondern bei dieser, hinter dem Befehl angegebenen Zeile gestartet.

So, das war zunächst alles, was Sie an Informationen zum Speichern und Laden mit dem Kassettenrecorder benötigen. Im Kapitel 'DAS KASSETTENLAUFWERK' wird an späterer Stelle auf alle Möglichkeiten eingegangen, die dieses Laufwerk bietet.

LÖSCHEN EINES PROGRAMMS

Wie bereits bekannt, wird das Programm beim Ausschalten des Rechners gelöscht. Doch dies ist sicherlich nicht die einzige Möglichkeit.
DER BEFEHL 'NEW' *LÖSCHT DAS IM SPEICHER BEFINDLICHE PROGRAMM.*

Geben Sie nun den Befehl 'NEW' ein, so wird das Programm gelöscht. Man sollte natürlich das zu löschende Programm vorher abspeichern.

Für alle skeptischen Leser: Wenn Sie nach dem 'NEW' den Befehl 'LIST' eingegeben, so werden Sie feststellen, daß das Programm wirklich gelöscht wurde. Es werden keine Zeilen mehr angezeigt. Auch der Befehl 'RUN' ist nun wirkungslos.

Sie haben nun die ersten Grundlagen zur Erstellung eines Programms erworben, was für Sie der Grundstein zum Erarbeiten der anschließenden, etwas anspruchsvolleren Kapitel sein wird. Wenn Sie in diesem Kapitel etwas überfordert wurden, so empfiehlt es sich, dieses Kapitel nochmals durchzuarbeiten.

KAPITEL 4: BASIC-EINFÜHRUNG

Dieses Kapitel soll eine Einführung in die Programmiersprache BASIC darstellen. Dabei werden nicht alle BASIC-Befehle stur nacheinander beschrieben, sondern es wird eine Adressenverwaltung Schritt für Schritt aufgebaut, in der die Befehle dann zum gegebenen Zeitpunkt beschrieben und eingesetzt werden. Der Leser dieses Buches soll durch dieses Kapitel nicht zum perfekten Programmierer werden, sondern einen intensiven Einblick in die praxisnahe Programmierung erhalten. Dieser Einblick bietet ihm dann beste Voraussetzungen, sich durch weitere Fachbücher weiterzubilden. Die Befehle, die in diesem Buch nicht behandelt wurden, finden Sie in Ihrem Handbuch.

PROBLEMBESCHREIBUNG ZUR ADRESSENVERWALTUNG

Eine Adressenverwaltung ist eines der beliebtesten Programme auf dem Heimcomputer. Die breite Einsatzfähigkeit trägt wesentlich dazu bei. Es gibt kaum jemand, der dieses Programm nicht einzusetzen weiß. Zwar ist eine Adressenverwaltung mit nur wenig Adressen per Heimcomputer nicht unbedingt effektiver als die herkömmliche Methode mit dem Adressenbüchlein, aber erheblich eindrucksvoller. Wenn sich jedoch sehr viele Adressen ansammeln, so sind sie in einem Heimcomputer bestens aufgehoben.

Welche Möglichkeiten sollte ein solches Programm bieten? Nun, auf jeden Fall müssen Adressen ein- und ausgegeben werden.

Bevor wir weitere Ansprüche an das Programm stellen, muß noch etwas klargestellt werden. In einem Programm, das Daten verwaltet, unterscheidet man zwischen zwei grundsätzlichen Bestandteilen:

1. Das Programm
2. Die Daten

Die Daten sind nicht Bestandteil des Programms. Dies wäre zwar möglich, aber diese Daten können dann nur vom Programmierer und nicht vom Anwender verwaltet werden. Eine Änderung der Daten hätte gleichzeitig eine Änderung des Programms zur Folge, was wir sofort wieder vegessen können.

Das Programm befindet sich also auf dem externen Speichermedium (Kassette oder Diskette) und wird bei Bedarf in den Rechner eingeladen. Aber was ist nun mit den Daten? Sie fallen erstmalig bei der Ersterfassung an. Wohin dann damit?

Wir müssen eine Datei organisieren. Eine Datei ist eine Ansammlung von Daten auf einem externen Speichermedium. Es gibt zwar mehrere Dateiorganisationsformen, wir beschäftigen uns aber nur mit der einfachsten Methode, der sequentiellen Datei. Sequentiell bedeutet, daß die Daten hintereinander angeordnet sind. Die zwei wichtigen Bestandteile unserer Adressenverwaltung sind nun

1. Das Programm
2. Die Datei

Dateiorganisation

Wenn ich jetzt fragen würde, welcher Teil zuerst organisiert werden muß, werden viele antworten: "Das Programm". Die Antwort ist aber falsch. Zuerst muß man sich im klaren sein, was wie wo abgespeichert wird. Über das "wie" sind wir uns im klaren, sequentiell soll gespeichert werden. Das "wo" ist noch nicht geklärt worden. Es gibt zwei Möglichkeiten:

1. Speicherung auf Kassette
2. Speicherung auf Diskette

Wir wollen unsere Daten auf dem Kassettenlaufwerk abspeichern. Im Anhang finden Sie die Programmänderungen zur Speicherung der Daten auf Diskette. Ändern Sie bitte die entsprechenden Teile des Programms zum Speichern auf einer Diskette gegebenenfalls um.

Bleibt nun noch die Frage, was gespeichert werden soll. "Adressen natürlich" werden viele spontan reagieren, doch sind wir uns überhaupt im klaren, was alles zu einer Adresse gehören soll? Natürlich nicht. Sammeln wir doch einmal alles, woraus sich eine Adresse zusammensetzen könnte:

-Anrede	*-Postleitzahl*
-Vorname	*-Ort*
-Name	*-Telefonnummer*
-Strasse	*-Bemerkung*

Diese einzelnen Bestandteile einer Adresse nennt man *DATENFELD*. Man spricht so z.B. vom Feld "Anrede". Die Gesamtheit der Datenfelder ergibt den *DATENSATZ*. Jeder Datensatz besteht also in unserem Beispiel aus diesen 8 Feldern. Alle Datensätze zusammen ergeben schließlich die Datei. Sehen wir uns diese Struktur einmal in folgendem Bild an:

```
FELD 1  FELD 2  FELD 3  ......  FELD n      DATENSATZ 1

FELD 1  FELD 2  FELD 3  ......  FELD n      DATENSATZ 2

FELD 1  FELD 2  FELD 3  ......  FELD n      DATENSATZ 3

FELD 1  FELD 2  FELD 3  ......  FELD n      DATENSATZ 4

FELD 1  FELD 2  FELD 3  ......  FELD n      DATENSATZ n

                      D A T E I
```

Die Hierarchie ist also DATEI - DATENSATZ - DATENFELD

RECHNERINTERNE SPEICHERUNG DER DATEN

Mit einer sequentiellen Datei kann man zwar sehr bequem arbeiten, jedoch gibt es hier auch Nachteile gegenüber anderen Organisationsformen. Stellen Sie sich vor, die gesamte Datei mit den Adressen befindet sich auf einer Kassette. Sie möchten nun eine ganz bestimmte Adresse aus dieser Datei haben. Hier tritt dann das Problem auf. Sie können nicht aus einer Datei nur einen Datensatz lesen. Der Kassettenrekorder und auch die Floppystation sind nicht in der Lage, aus einer sequentiellen Datei einzelne Datensätz zu lesen. Wie kann man nun auf die Adressen zugreifen?

Um auf einen Datensatz zuzugreifen, muß die gesamte Datei in den Rechner eingelesen werden. Eine sequentielle Datei darf demnach nicht größer als der vorhandene Speicherplatz des Rechners sein. Ein Vorteil stellt sich jedoch wieder heraus. Wenn die Datei einmal eingelesen ist, so kann im Rechner blitzschnell auf die Datensätze zugegriffen werden. Für den gesamten Programmablauf gilt also folgende Regelung:

1. Datei laden
2. Datensätze lesen, ändern, löschen .....
3. Datei speichern

Nach dem Start des Programms muß also zuerst die Adressendatei komplett eingelesen werden. Danach können die Datensätze verarbeitet werden. Vor Beendigung des Programms muß die Datei jedoch wieder gespeichert werden, sofern sie geändert wurde. Wurden die Datensätze jedoch nicht geändert und auch keine gelöscht oder hinzugefügt, so stimmt die Datei noch mit der ursprünglich eingeladenen Datei überein. Sie braucht also nicht wieder gespeichert zu werden.

VARIABLEN

Wie bekannt, werden die Datensätze im Rechner gespeichert. Die Frage ist nur, wo und wie werden sie gespeichert?

RECHNERINTERNE DATEN WERDEN IN VARIABLEN GESPEICHERT.

Variablen sind also das Zauberwort. Es sind Speicherbereiche im ATARI, die mit einem Namen versehen werden. Durch Angabe dieses Namens kann auf diese Speicherbereiche zugegriffen werden. Wir haben schon zwei verschiedene Datentypen kennengelernt. Da gibt es numerische Daten (Zahlen) und alphanumerische Daten (Strings, Text). Stringvariablen werden durch das Zeichen '$' hinter der Variablenbezeichnung gekennzeichnet. Woraus setzen sich nun die Bezeichnungen der Variablen zusammen? Nun, zunächst darf der Variablenname beliebig lang sein, wenn der Speicherplatz es zuläßt. Das erste Zeichen **muß** ein Buchstabe sein, weitere Zeichen dürfen Zahlen oder Buchstaben sein. Es gibt wenige Ausnahmen. BASIC-Befehle oder Funktionen dürfen nicht verwendet werden:

z.B. PRINT, RUN, NEW

Kein Variablenname darf mit einem BASIC-Wort beginnen. Ein Verzeichnis sämtlicher reservierten BASIC-Worte finden Sie im Anhang.

Beispiele numerische Variablen:

ZAEHLER
PLZ
TELEFON
BETRAG

Beispiele Stringvariablen:

ANREDE$
ARTIKELBEZEICHNUNG$
KONTOBEZEICHNUNG$
X$

Bei der Auswahl der Variablen sollten Sie sinnvolle Bezeichnungen verwenden. So wird die Variable zum Speichern des Vornamens z.B. *VORNAME$* genannt. Den Einkaufpreis würde man *EINKAUFSPR* nennen, usw. Sinnvolle Variablennamen tragen wesentlich zur Übersichtlichkeit des Programms bei.

VARIABLENVERARBEITUNG

Wenden wir uns nun wieder nach langer Zeit dem Rechner zu. Wir werden nun Variablen einrichten, verarbeiten und ausgeben. Die Speicherung von numerischen Daten in Variablen z.B. ist eigentlich ein Kinderspiel. Nehmen wir an, die Zahl 45.12 soll in der Variablen *EINKAUFSPR* gespeichert werden. Der entsprechende Befehl ist dann

EINKAUFSPR=45.12

Ist doch einfach, oder? Soll diese Variable ausgegeben werden, so lautet der Befehl

PRINT EINKAUFSPR

Geben Sie beide Befehle nun einmal in den Rechner ein. Soll die Variable *EINKAUFSPR* wieder gelöscht werden, so geben Sie sinngemäß den Befehl

EINKAUFSPR=0

ein. Die Variablen können aber auch für Rechenoperationen benutzt werden. Geben wir einmal den doppelten Wert von *EINKAUFSPR* aus. Wenn Sie die Variable gelöscht haben, so geben Sie bitte erneut '*EINKAUFSPR=45.12*' ein.

*PRINT EINKAUFSPR*2*

Der Rechner verdoppelt *EINKAUFSPR* und gibt das Ergebnis (90.24) aus. Die Variable selbst behält den Wert 45.12. Was aber ist zu tun, wenn die Variable *EINKAUFSPR* verdoppelt werden soll? Da die Variable einen neuen Wert erhalten soll, beginnt der entsprechende Befehl mit '*EINKAUFSPR=*'. Der gesamte Befehl zur Verdoppelung ist:

*EINKAUFSPR=EINKAUFSPR*2*

Es wird immer erst der Wert rechts vom Gleichheitszeichen errechnet und dann in die Variable links vom Gleichheitszeichen abgelegt.

STRINGVARIABLEN

Die Verwaltung der Stringvariablen ist beim ATARI grundsätzlich anders als bei den meisten anderen Rechnern. Stringvariablen werden auf ähnliche Art und Weise wie die numerischen Variablen eingerichtet. Es muß jedoch zunächst Speicherplatz für die Variable frei gehalten werden. Der folgende Befehl erledigt diese Aufgabe:

```
Problem:     Speicherplatz für Strings reservieren

Befehl:      DIM str(n)

Parameter:   str - Stringvariable
             n -   Länge des Strings (max. Länge ist
                   nur durch Speicherkapazität be-
                   grenzt
```

Beispiel:	DIM FELD$(20) Es werden 20 Zeichen für die Stringvariable FELD$ reserviert
Bemerkung:	Es können mehrere Variablen mit einem DIM eingerichtet werden. Ein Beispiel: DIM A$(10),B$(20),FELD$(12) Werden Stringvariablen nicht eingerichtet, bevor man mit ihnen arbeitet, oder werden bestehende Strings nochmals eingerichtet, so wird 'ERROR-9' gemeldet

Speichern wir nun den String "*FRANKFURT*" in der Stringvariablen *STADT$*. Die Befehle dazu lauten

DIM STADT$(20)
STADT$="FRANKFURT"

Wir haben zunächst die Stringvariable mit 20 Zeichen eingerichtet, um anschließend "*FRANKFURT*" zu speichern.

Auch Stringvariablen können mit dem Befehl *PRINT* auf dem Bildschirm ausgegeben werden. Sehen Sie selbst:

PRINT STADT$

Was geschieht? Natürlich, der in STADT$ gespeicherte String wird ausgegeben.

TEILSTRINGS

```
Problem:      Teilstrings bilden

Funktion :    str(von,bis)

Parameter:    str - Stringvariable
              von - Position, ab der der Teilstring
                    gebildet werden soll
              bis - Position, bis der der Teilstring
                    gebildet werden soll

Beispiel:     PRINT STADT$(2,5)
              Gibt den Teil des Strings STADT$ aus,
              der ab der Position 2 bis zur Positio 5
              liegt

Bemerkung:    Bei falschen Parameter 'von' und 'bis'
              ('von' größer als 'bis', oder 'von'=0)
              erscheint der Fehler ERROR-5
```

Wollen wir nun aus unserem *STADT$* die ersten 5 Zeichen ausgeben, so benötigen wir den folgenden Befehl:

PRINT STADT$(1,5)

So wie Teile aus einem String entnommen werden können, ist es auch möglich, Teile des Strings zu überschreiben. Ein Beispiel: Die ersten fünf Zeichen von STADT$ sollen mit Leerzeichen überschrieben werden. Dazu geben wir den folgenden Befehl ein:

STADT$(1,5)="   "

Ist die Zuweisung länger als der Teilstring, so wird die Zuweisung "abgeschnitten". Eine Befehlsfoge, die dies bestätigt:

*STADT$(1,5)="**********"*
PRINT STADT$

Hier sehen Sie, daß nur die ersten fünf Zeichen der Zuweisung übernommen wurden.

Ist die Zuweisung kürzer, als der angegebene Teilstring, so wird die Zuweisung bis zum letzten Zeichen übernommen. Auch hier ein Beispiel:

STADT$(1,5)="--"
PRINT STADT$

Es wurden nur die beiden Striche übernommen, obwohl der Teilstring eigentlich größer angegeben wurde.

ENTFERNEN DER STRINGVARIABLEN

Alle Variablen speichert der Rechner im sogenannten Variablenbereich. Dieser Bereich nimmt maximal 128 verschiedene Variablen auf und kann mit zwei Befehlen gelöscht werden:

NEW	*löscht Programm und Variablen*
RUN	*löscht Variablen vor Programmstart*
CLR	*löscht nur die Variablen*

Sie haben also wieder einen neuen Befehl kennengelernt, den *CLR*. Doch wozu wird dieser Befehl benötigt? Wenn Sie z.B. ein Programm laufen lassen, in dem Stringvariablen eingerichtet wurden, dieses unterbrechen und anschließend anstatt mit 'RUN' mit dem Befehl *GOTO* starten, so wird versucht, die Variablen im Programmablauf erneut einzurichten. Dies führt zum *ERROR-9*. Wenn jedoch vorher mit *CLR* gelöscht wird, so hat der Einsprung mit *GOTO* keine unangenehmen Folgen.

Soll eine Stringvariable wieder gelöscht, jedoch nicht aus dem Variablenbereich entfernt werden, so wird der Variablen einfach ein Leerstring zugewiesen. Löschen wir nun auf diese Weise die Variable *STADT$*:

STADT$=""

Ein Leerstring besteht also aus zwei unmittelbar aufeinanderfolgenden Anführungszeichen. Selbstverständßlich kann mit diesen Strings nicht gerechnet werden. Auch nicht, wenn eine Zahl als String abgelegt wird. Das heißt, wenn Sie z.B. *X$="123"* eingeben würden, so können Sie trotzdem nicht ohne weiteres mit *X$* rechnen.

Strings können auch verkettet werden. Die folgende Befehlsfolge bestätigt es:

```
DIM STADT$(30),STADT1$(15),STADT2$15)
STADT1$="DUESSELDORF"
STADT2$="FRANKFURT"
STADT$(1,15)=STADT1$
STADT$(16,30)=STADT2$
PRINT STADT$
```

Zuerst wurden die drei Variablen eingerichtet. Dann wurden die beiden Städte in *STADT1$* und *STADT2$* gespeichert. Danach wurden die beiden Stringvariablen hintereinander in dem String STADT$ übernommen. Der vierte Befehl hat den String *STADT$* dann ausgegeben.

Wir haben jetzt die zwei wichtigsten Variablentypen kennengelernt. Die numerische Variable und die Stringvariable, die durch ein zusätzliches '$' gekennzeichnet ist.

TABELLEN

Tabellen werden in der Datenverarbeitung oft eingesetzt. Wenn Sie z.B. mehrere Datensätze einer Datei im Rechner speichern wollen, so ist es sehr umständlich, jedem Datensatz einen Variablennamen zu geben. In diesem Fall wird die Datei, vorrausgesetzt sie passt in den Speicher, in eine Tabelle eingelesen. Mittels einer Zahl (Index) kann auf jeden Datensatz in dieser Tabelle zugegriffen werden. Dieser Index ist die relative Satznummer vom Tabellenanfang an gerechnet.

Da der ATARI keine Stringvariablen indizieren kann, muß eine andere Technik angewandt werden. Datenelemente der Tabelle werden alle in einer Stringvariablen abgelegt. Dabei muß die Länge der Elemente immer gleich sein. Lassen Sie uns hier ein Beispiel zur Verdeutlichung heranziehen:

Es sollen fünf Städte in einer Tabelle mit dem Namen TABELLE$ gespeichert werden:

BONN
PARIS
LONDON
ROM
MADRID

Bei der einheitlichen Elementlänge richten wir uns nach dem längsten Städtenamen. Jedes Element soll also 6 Zeichen in Anspruch nehmen. Ist der Städtename kleiner als 6, so wird der Rest mit Leerzeichen aufgefüllt.

Wir müssen also 5 Elemente mit jeweils 6 Zeichen in dem String unterbringen. Dazu reservieren wir 30 Zeichen. Drücken Sie nun die RESET-Taste am Rechner und geben den ersten Teil dieses Beispielprogramms ein:

```
100 DIM TABELLE$(30)
```

Diesen String wollen wir nun mit Leerzeichen füllen. Ein kleiner Trick, den Sie sich merken sollten, erfüllt diese Aufgabe:

```
110 TABELLE$(1)=" "
120 TABELLE$(30)=" "
130 TABELLE$(2)=TABELLE$
```

Es wird hier wie folgt vorgegangen:

1. An der ersten und letzten Position des Strings wird ein Leerzeichen gespeichert.

2. Der zweiten Position des Strings wird der gesamte String zugewiesen.

Nun haben wir die Tabelle zur Aufnahme der Städtenamen vorbereitet. Die Städtenamen sollen, wie in der folgenden Skizze verdeutlicht, im String *TABELLE$* gespeichert werden:

```
         1         2         3
123456789012345678901234567890

BONN  PARIS LONDONROM   MADRID
```

Die folgenden Zeilen speichern die Namen an diese Positionen:

```
140 TABELLE$(1,6)="BONN"
150 TABELLE$(7,12)="PARIS"
150 TABELLE$(13,18)="LONDON"
160 TABELLE$(19,24)="ROM"
170 TABELLE$(25,30)="MADRID"
180 PRINT TABELLE$
```

Die Zeile 180 gibt die gesamte Tabelle auf dem Bildschirm aus. Hier sehen Sie, wie diese Tabelle nun organisiert ist.
Der Sinn und Zweck dieser Tabellenverarbeitung ist, jeden Tabellenplatz mit der relativen Elementnummer - mit dem Index - zu selektieren. Dazu ist ein Algorithmus notwendig. Angenommen, der Index ist in der Variablen *INDEX* gespeichert. Dann gibt der folgende Befehl den entprechenden Tabellenplatz aus:

INDEX=3
*PRINT TABELLE$(INDEX*6-5,INDEX*6)*

Probieren Sie diesen Algorithmus nun aus, indem Sie den *INDEX* jeweils ändern und den entsprechenden Tabellenplatz mit dem angegebenen *PRINT* ausgeben.

Rechnen wir dieses Beispiel nun einmal mit dem Index 3 aus. Die Anfangsposition dieses Platzes berechnet sich aus *'INDEX*6-5'* (3*6-5). Die Anfangsposition ist also 13. Die Endposition ergibt sicht aus *'INDEX*6'*, ist also 18. Es wird also stets der gewünschte Tabellenplatz ausgegeben. Dieser Algorithmus kann verallgemeinert werden:

```
Berechnung der Anfangs- und Endposition
eines eindimensionalen Tabellenplatzes

ANFPOS = INDEX * LAENGE - (LAENGE-1)
ENDPOS = INDEX * LAENGE

INDEX - relative Tabellenplatznummer
LAENGE - Länge eines Tabellenplatzes
```

Die soeben erstellte Tabelle bezeichnet man als eindimensional, da sie nur einen Index besitzt. Nun können Tabellen aber mit beliebig vielen Indizes gestaltet werden. Anstatt der zuvor aufgebauten Tabelle kann auch eine zweidimensionale Tabelle erstellt werden. Wir bauen also 6 Datensätze mit jeweils zwei Datenfeldern auf. Wir müssen für jedes Feld 10 Zeichen reservieren, da das längste Feld in dieser Tabelle 10 Zeichen lang ist (*FRANKREICH*). Das erste Datenfeld ist die Hauptstadt und das zweite Feld das entsprechende Land, jeweils mit einer Länge von 10 Zeichen. Sehen Sie sich zunächst die Struktur dieser zweidimensionalen Tabelle an:

	Feld 1	*Feld 2*
Satz 1	*BONN*	*BRD*
Satz 2	*PARIS*	*FRANKREICH*
Satz 3	*LONDON*	*ENGLAND*
Satz 4	*ROM*	*ITALIEN*
Satz 5	*MADRID*	*SPANIEN*

Die zweidimensionale Tabelle, realisiert mit der Stringvariablen *TABELLE$*, muß also 5 x 20, also 100 Zeichen umfassen. Richten wir nun diese Tabelle ein, nachdem das alte Programm mit NEW gelöscht wurde.

```
100 DIM TABELLE$(100)
110 TABELLE$(1)=" "
120 TABELLE$(100)=" "
130 TABELLE$(2)=TABELLE$
```

Nun werden nacheinander die Sätze in die Tabelle eingetragen.

```
140 TABELLE$(1,10)="BONN":TABELLE$(11,20)="BRD"
150 TABELLE$(21,30)="PARIS":TABELLE$(31,40)="FRANKREICH"
160 TABELLE$(41,50)="LONDON":TABELLE$(51,60)="ENGLAND"
170 TABELLE$(61,70)="ROM":TABELLE$(71,80)="ITALIEN"
180 TABELLE$(81,90)="MADRID":TABELLE$(91,100)="SPANIEN"
```

Diese Eingabe ist mühselig, jedoch zum Verständnis der zweidimensionalen Tabelle im ATARI unerläßlich.

Nun haben wir die Tabelle vorliegen und müssen nur noch eine Technik entwickeln, auf die einzelnen Felder zuzugreifen. Wir benötigen dazu zwei Indizes. Der erste Index bezeichnet die Satznummer und der zweite die Feldnummer in diesem Satz. Um z.B. "*ROM*" aus der Tabelle zu lesen, muß der erste Index aufvier

und der zweite auf eins gesetzt werden. Wir verwenden die Variablen *INDEX1* und *INDEX2*. Auch hier wird wieder ein Algorithmus benötigt:

```
140 INDEX1=4:INDEX2=1
150 ANFPOS=(INDEX1*20-19)+(INDEX2-1)*10
160 ENDPOS=ANFPOS+9
170 PRINT TABELLE$(ANFPOS,ENDPOS)
```

Probieren Sie dieses Programm nun mehrmals aus, und ändern Sie jeweils die Indizes in Zeile 140.

Auch für die zweidimensionale Tabelle gibt es einen allgemeinen Algorithmus:

```
        Berechnen der Anfangs- und Endposition
        eines zweidimensionalen Tabellenplatzes

ANFPOS = INDEX1 * SATZ-(SATZLEN-1)+(INDEX2-1)*FELD
ENDPOS = ANFPOS+FELD-1

INDEX1 - Satzindex
INDEX2 - Feldindex
SATZ   - Länge eines Satzes der Tabelle
FELD   - Länge eines Feldes der Tabelle
```

Kommen wir nun wieder zu unserer Adressenverwaltung zurück. Eine solche zweidimensionale Tabelle ist ideal für die Aufnahme der sequentiellen Adressendatei. Der erste Index numeriert die Datensätze, der zweite Index numeriert die Datenfelder innerhalb der Datensätze. Da unsere Datensätze 7 Felder enthalten, ist der zweite Index maximal 7. Der erste Index, also die Anzahl der Adressen, ist wahlweise. Legen wir uns hier auf maximal 100 Adressen fest. Die Feldlänge soll 20 Zeichen betragen. Wir benötigen also eine Tabelle von 7 * 20 * 100 Zeichen. 14000 Zeichen des Speicherplatzes werden also von dieser Tabelle belegt.

Zum Abschluß ein Tip für Benutzer des ATARI 600XL ohne Speichererweiterung: Zum Einsatz der Adressenverwaltung wird Ihr Speicher sicherlich nicht ausreichen, da die Adressen auf vielleicht 10 begrenzt werden müssen. Ich hoffe, daß Sie trotzdem genau so viel Spaß an diesem Programm haben werden, wie die Anwender der "großen Brüder" ATARI 800XL und 130 XE. Sicher werden auch Sie bald Ihren Speicher aufrüsten. Besitzer des 130 XE können trotz des größeren Speichers von BASIC aus nur die üblichen 64 KByte ausnutzen.

DATENEINGABE ÜBER TASTATUR

In den bisherigen Übungen haben wir nur Daten verarbeitet und ausgegeben. Interessant wird es erst, wenn dem Programm Daten über die Tastatur übergeben werden. Der Befehl dazu heißt *INPUT*. Doch *INPUT* alleine hat keine Wirkung. Dem Befehl folgt ein Parameter: Eine Variable, in der die eingegebenen Daten gespeichert werden sollen. Wichtig: Der Befehl *INPUT* kann nicht im Direktmodus, sondern nur im Programm eingesetzt werden. Doch beachten Sie zunächst die Befehlsbeschreibung:

```
Problem:     Dateneingabe über Tastatur

Befehl:      INPUT var

Parameter:   var     - Variable, in der die Eingabe
                       gespeichert wird.

Beispiel:    INPUT X
             Es wird ein numerischer Wert von der
             Tastatur eingelesen und in der Variablen
             X gespeichert.

Bemerkung:   Der Befehl kann auch im Direktmodus
             eingegegen werden.
```

Geben Sie nun den Befehl *NEW* ein, der ein zuvor im Speicher befindliches Programm löscht. Die folgenden BASIC-Zeilen demonstrieren den Einsatz des Befehls *INPUT*:

```
10 PRINT "KREISBERECHNUNG"
20 PRINT "---------------"
30 PRINT "DURCHMESSER : ";
40 INPUT DURCHM
50 PRINT "KREISFLAECHE: ";(DURCHM/2)^2*3.14
```

Starten Sie dieses Programm mit dem Befehl *RUN*. In Zeile 40 wird der Durchmesser des zu berechnenden Kreises abgefragt. Die Eingabe wird stets mit *RETURN* abgeschlossen. Hinter dem String nach dem PRINT muß ein Semikolon angegeben werden, damit die Eingabe des folgenden Wertes direkt hinter dem erklärenden Text erfolgt.

Strings können ebenfalls über die Tastatur mit dem Befehl INPUT eingelesen werden. Auch hier zunächst ein Beispiel. Löschen Sie zuvor das alte Programm mit *NEW*.

```
5 DIM NAME$(20)
10 PRINT "WIE HEISSEN SIE? ";
20 INPUT NAME$
30 PRINT "GUTEN TAG ";NAME$;", WIE GEHT ES IHNEN?"
```

Diese Beispiele demonstrieren alles, was der Befehl *INPUT* zu bieten hat. Beachten Sie, daß das Programm auch weiterläuft, wenn nichts eingegeben, also nur *RETURN* gedrückt wird. Dies gilt jedoch nur für Strings, nicht für numerische Variablen. Wird der INPUT einer numerischen Variable mit *RETURN* abgeschlossen, ohne daß eine gültige Zahl eingegeben wurde, so wird '*ERROR-8*' gemeldet. Die Eingabe von Strings jedoch kann ohne weiteres beendet werden. Der entsprechende String wird dann gelöscht.

SCHLEIFEN

Lassen Sie uns zunächst eine Übung machen, die dann später zur Schleifensteuerung führt. Es sollen fünf Namen über die Tastatur eingelesen werden. Diese Namen sind jeweils maximal 10 Zeichen lang und sollen in der Tabelle *NAME$* gespeichert werden. Mit den bisher beschriebenen Befehlen würden wir folgendes Programm aufbauen:

```
100 DIM NAME$(50),N$(10)
110 NAME$(1)=" ":NAME$(50)=" ":NAME$(2)=NAME$
120 PRINT"EINGABE DER NAMEN"
130 PRINT"-----------------"
140 INPUT N$:NAME$(1,10)=N$
150 INPUT N$:NAME$(11,20)=N$
160 INPUT N$:NAME$(21,30)=N$
170 INPUT N$:NAME$(31,40)=N$
180 INPUT N$:NAME$(41,50)=N$
```

Mit diesem Programm werden jetzt nacheinander die fünf Namen eingelesen und an entsprechender Stelle gespeichert. Dazu werden in der Zeile 100 zwei Variablen eingerichtet: *NAME$* als Tabelle zur Speicherung der fünf Namen und *N$* zur Aufnahme des mit *INPUT* einzulesenden Namens. Wie üblich wird die komplette Tabelle (*NAME$*) mit Leerzeichen aufgefüllt. Erinnern Sie sich an die Vorgehensweise zum Auffüllen einer Tabelle mit einem bestimmten Zeichen? Wenn nicht, folgt noch einmal eine kurze Beschreibung, die Sie sich unbedingt einprägen sollten:

AUFFÜLLEN EINES STRINGS MIT EINEM ZEICHEN

1. Füllzeichen an ERSTE Position des Strings setzen (z.B. T$(1)="*")

2. Füllzeichen an LETZTE Position des Strings setzen (z.B. T$(100)="*")

3. Gesamten String nach ZWEITE Stelle des Strings speichern (z.B. T$(2)=T$)

MERKE: ERSTES - LETZTES - ZWEITES ZEICHEN

Sicher werden Sie diesen Vorgang nun endgültig Ihrem Gedächtnis übergeben haben. Dort ist noch sehr viel Platz für die vielen Informationen, die auf den folgenden Seiten auf Sie zukommen.

Nachdem die Tabelle nun mit Leerzeichen aufgefüllt wurde, ist sie für die folgenden fünf Namen aufnahmebereit. Dazu lesen wir den jeweiligen Namen in einem Zwischenspeicher (*N$*) ab, der dann zur entsprechenden Stelle in der Tabelle gespeichert wird. Dieser Vorgang wiederholt sich dann bis zum letzten einzulesenden Namen. Die Zeilen 140 bis 180 sind fast identisch. Einem erfahrenen Programmierer würden beim Anblick dieses Programmierstils die Haare zu Berge steigen. Doch er kann sie bald wieder fallen lassen, denn nun lernen Sie eine Methode kennen, sich oft wiederholende Vorgänge elegant zu einem gezielt gesteuerten Vorgang zu komprimieren.

Sie bemerken, daß sich die Zeilen 140 bis 180 nur in den Anfangs- und Endpositionen der Tabellenplätze unterscheiden. Sie werden bei jeder Anweisung um zehn erhöht. Um diese Wiederholungem zu vermeiden, bedient sich der Programmierer einer unverzichtbaren Programmiertechnik, der Schleifensteuerung. Schauen Sie sich zunächst den Befehl an, der die Schleife einleitet:

Problem:	Schleifensteuerung
Befehl:	FOR var = anf 1 TO end STEP ink
Parameter:	var - numerische Variable anf - Anfangswert von 'var' end - Endwert von 'var' ink - Inkrement, konstanter Wert (keine Variable, um den die Variable 'var' nach jedem Schleifendurchlauf erhöht wird
Beispiel:	FOR A=1 TO 20 STEP 1 Die Variable A wird nach jedem Durchlauf um 1 erhöht, bis sie den Wert 20 erhält (20 Schleifendurchläufe)
Bemerkung:	Ist das Inkrement 1, so kann der Parameter STEP weggelassen werden. Der Befehl aus dem Beispiel könnte demnach auch so eingegeben werden: FOR A=1 TO 20

Jede Schleife beginnt mit einer *FOR*-Anweisung. Alle anschließenden Befehle gehören dann zu dieser Schleife und werden demnach mehrmals ausgeführt. Doch wie wird das Ende der Schleife gekennzeichnet? Dafür gibt es eine weitere Anweisung, die die Aufgabe hat, die Schleifenvariable auf den Endwert zu prüfen. Ist der Endwert noch nicht erreicht, so wird die Schleifenvariable um das Inkrement erhöht und die Schleife nochmals durchlaufen. Entspricht der Endwert jedoch der Schleifenvariablen, oder ist er sogar größer, so ist die Schleife beendet und das Programm wird hinter der Schleife fortgeführt.

Problem:	Schleifenende festlegen
Befehl:	NEXT var
Parameter:	var - Schleifenvariable
Beispiel:	NEXT A Kennzeichnet das Ende der im vorherigen Beispiel enthaltenen Schleife.
Bemerkung:	Ein NEXT-Befehl ohne entsprechenden FOR-Befehl erzeugt einen ERROR-13

Mit diesen beiden Befehlen können wir nun beliebige Schleifen programmieren. Wollen Sie z.B. die Zahlen 1 bis 10 nacheinander auf dem Bildschirm ausgeben, so verwenden Sie die folgende Schleife:

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
```

Bitte beachten Sie, daß Sie dieses Programm nur eingeben können, wenn Sie das im Speicher vorhandene Programm vorher sichern. Sie können diese Schleife jedoch auch im Direkt-Modus, das heißt ohne Zeilennummern eingeben. Dazu geben Sie die drei Anweisungen, getrennt mit einem Doppellpunkt, ohne Zeilennummer ein und drücken die Taste *RETURN*.

FOR I=1 TO 10:PRINT I:NEXT I

Ein weiteres Beispiel: Die Zahlen von 1 bis 10 sollen addiert und das Ergebnis ausgegeben werden:

FOR I=1 TO 10:A=A+I:NEXT I:PRINT A

Solche Multi-Anweisungen sollten Sie in Programmen nach Möglichkeit vermeiden, da dies der Übersichtlichkeit des Programms schadet.

Wenden wir uns wieder der anfangs beschriebenen Dateneingabe mit *INPUT* zu. Hier sollten fünf Namen in eine Tabelle eingelesen werden. Wenn wir nun eine Schleife einsetzen, so benötigen wir nur eine *INPUT*-Anweisung, in der die Anfangs- und Endposition mit Hilfe der Schleifenvariablen berechnet wird. Die Schleife soll von 1 bis 41 in 10er Schritten laufen. Wie sieht nun der entsprechende *FOR*-Befehl mit allen Parametern aus?

FOR I=1 TO 41 STEP 10

Das gesamte, geänderte Programm ist folgendes:

```
100 DIM NAME$(50),N$(10)
110 NAME$(1)=" ":NAME$(50)=" ":NAME$(2)=NAME$
120 PRINT "EINGABE DER NAMEN"
130 PRINT "-----------------"
140 FOR INDEX=1 TO 41 STEP 10
150 INPUT N$:NAME$(INDEX,INDEX+9)
160 NEXT INDEX
```

Eine sehr elegante Lösung zum Einlesen von 5 Strings, oder? Interessant ist der genaue Ablauf nach dem Starten dieses Programms. Zuerst werden die beiden *PRINT*-Befehle ausgeführt. Nun beginnt die Schleife mit der Variablen *INDEX*, dem Anfangswert 1, dem Endwert 41 und dem Inkrement (Erhöhungswert) 10. Erinnern Sie sich, wird kein *STEP*-Parameter angegeben, so wird mit dem Inkrement 1 gearbeitet. Zu Anfang der Schleife erhält *INDEX* den Wert 1. In Zeile 140 wird also zunächst der Name in den Zwischenspeicher *N$* eingelesen und anschließend zum entsprechenden Tabellenplatz gebracht. Die Schleifenvariable enthält jeweils die Anfangsposition des Tabellenplatzes (1, 11, 21, 31, 41). Der Endwert errechnet sich immer aus *INDEX+9* (10,20,30,40,50). Anschließend prüft der Befehl *NEXT*, ob das Schleifenende (Variable *INDEX* gleich oder größer 41) erreicht ist. Da dies nicht der Fall ist, wird *INDEX* um das Inkrement 1 erhöht und das Programm hinter dem Befehl

FOR, also in Zeile 150, fortgesetzt. Die Variable *INDEX* enthält nun den Wert 11 und bestimmt die nächste Anfangsposition des Namens in der Tabelle. So geht es nun weiter, bis der Befehl *NEXT* die Schleife beendet. Da hinter dem Befehl *NEXT* keine Anweisung folgt, wird das Programm beendet.

Nehmen wir nun an, daß nach Eingabe der Namen alle nochmals ausgegeben werden. Dazu muß eine weitere Schleife angehängt werden:

```
170 FOR INDEX=1 TO 41 STEP 10
180 PRINT "NAME ";NAME$(INDEX,INDEX+9)
190 NEXT INDEX
```

Es ist also alles nicht so kompliziert, wie Sie es vielleicht zu Anfang angenommen haben.

Was kann man noch mit einer Schleife anfangen? Nun, z.B. können Sie eine Warteschleife aufbauen, die den Programmablauf verzögert. Hier bauen Sie nur eine Schleife mit *FOR* auf und schließen Sie unmittelbar dahinter mit *NEXT* ab. Nun ist es wichtig, wie oft so eine Schleife durchlaufen werden muß, um z.B. eine Warteschleife von 1 Sekunde zu erzeugen. Es kann davon ausgegangen werden, daß eine derartige Schleife 500 mal in der Sekunde durchlaufen wird. Eine Warteschleife von ca. 3 Sekunden wird dann wie folgt aufgebaut:

FOR X=1 TO 1500:NEXT X

Erinnern Sie sich: In einer Zeile können mehrere Befehle eingesetzt werden, wenn diese durch einen Doppelpunkt getrennt werden. Lassen Sie uns nun in unserem Programm eine Warteschleife zwischen der Dateneingabe und der Datenausgabe einfügen. Eine Zeilennummer, die dazwischen liegt, ist z.B. die Zeile 165. Wie sieht diese Zeile nun aus, wenn wir ca. 2 Sekunden verzögern möchten?

```
165 FOR X=1 TO 1000:NEXT X
```

Lassen Sie jetzt das Programm mit *LIST* auf dem Bildschirm anzeigen. Sie werden sehen, daß alle nachträglich eingegebenen Zeilen ordnungsgemäß eingesetzt wurden. Starten Sie das Programm nun mit *RUN*, dann werden Sie die Auswirkung der Zeile 165 bemerken.

Bisher haben wir nur mit einem positiven Inkrement gearbeitet, das durchaus auch negativ sein kann. Soll die Tabelle im vorherigen Beispiel nicht von vorne nach hinten, sondern umgekehrt gefüllt werden, so ist dies durchaus möglich. Der Anfangswert der Schleife ist dann 41, der Endwert 1 und das Inkrement, also der *STEP*-Parameter -10. Wie müssen die Zeilen 140 und 170 abgeändert werden? Sicher haben Sie die Lösung längst parat:

```
140 FOR INDEX=41 TO 1 STEP -10
170 FOR INDEX=41 TO 1 STEP -10
```

Lassen Sie uns noch ein paar Übungen zum Befehl FOR machen. Es sollen die FOR-Befehle zu folgenden Schleifen aufgebaut werden:

	Schleifen-variable	*Anfangs-wert*	*End-wert*	*Inkrement*
A)	INDEX	10	18	1
B)	ZAHL	30	15.5	-0.5
C)	SCHLEIFE	590	1800	0.25
D)	ZEIT	1	10	0.3

Die Lösungen:

A) FOR INDEX=10 TO 18 STEP 1
oder
FOR INDEX=10 TO 18

B) FOR ZAHL=30 TO 15.5 STEP -0.5

C) FOR SCHLEIFE=590 TO 1800 STEP 0.25

D) FOR ZEIT=1 TO 10 STEP 0.3

Noch eine Anmerkung zum Abschluß des Kapitels. Sollten Sie mit *FOR* Schleifen aufbauen, die logisch falsch sind (z.B. FOR I= 10 TO 0), so wird die Schleife grundsätzlich einmal durchlaufen!

ERSTE REAKTIONEN DES PROGRAMMS

Nachdem wir so weit in BASIC fortgeschritten sind, wird es Zeit, mit dem Adressenprogramm zu beginnen. Wie sollte man ein solches Programm anfangen? Wenn Sie schon einmal vergleichbare Programme bedient haben, wissen Sie sicher, was ein solches Programm nach dem Starten auf dem Bildschirm zeigt. Es meldet sich meist mit einem Programmkopf. Diesen Kopf, der nur aus wenigen PRINT-Anweisungen besteht, wollen wir zunächst erstellen. Anweisungen, die Ihnen gänzlich unbekannt sind, sollten Sie nicht daran hindern, trotzdem den Anfang des Programms einzugeben und zu starten. Diese Anweisungen werden danach beschrieben. Bitte sichern Sie gegebenenfalls das im Speicher befindliche Programm und drücken anschließend die RESET-Taste.

```
100 REM ====================
110 REM    PROGRAMMKOPF
120 REM ====================
130 PRINT CHR$(125)
140 FOR I=1 TO 38:PRINT "=";:NEXT I
150 POSITION 12,2
160 PRINT "ADRESSENVERWALTUNG"
170 FOR I=1 TO 38:PRINT "=";:NEXT I
```

Nachdem Sie diesen Teil des Programms eingegeben und gestartet haben, wollen Sie natürlich wissen, was die neuen Befehle bedeuten. Fangen wir bei den Zeilen 100-120 an. Hier wird dieser Abschnitt des Programms dokumentiert. Damit der Rechner den dokumentierenden Teil nicht als Befehl interpretiert, werden diese Zeilen mit *REM* gekennzeichnet. *REM* bedeutet *REMARKS*,

also Anmerkungen. Solche *REM*-Zeilen können Sie an beliebigen Stellen im Programm einfügen. Die Kennzeichnung der verschiedenen Programmabschnitte trägt zur Übersichtlichkeit des Programms bei.

Die nächste Zeile beinhaltet nichts, was Ihnen unbekannt ist. Hier wird 38 mal das Zeichen "=" ausgegeben, was den anschließenden Titel des Programms hervorheben soll. Wichtig ist hier das Semikolon hinter dem *PRINT*-Befehl, da damit die Zeichen hintereinander, nicht untereinander ausgegeben werden. Diese Zeile wird ebenfalls in 160 ausgegeben.

Der Befehl *PRINT* gibt Daten auf dem Bildschirm aus. An welcher Stelle des Bildschirms ausgegeben wird, hängt von dem letzten *PRINT*-Befehl ab. Um die Ausgabe mit *PRINT* an eine beliebige Stelle des Bildschirms einzuleiten, enthält Ihr BASIC den folgenden Befehl:

```
Problem:    Positionieren des Cursors

Befehl:     POSITION sp,z

Parameter:  sp - Spalte
            z  - Zeile

Beispiel:   POSITION 5,10
            setzt den Cursor auf die 5. Spalte der
            10. Zeile

Bemerkung:  Wird eine ungültige Spalten- oder Zeilen-
            Angabe gemacht, so wird ERROR-141 gemeldet
```

Nach jedem *PRINT*-Befehl wird die Position des Cursors verändert. Diesen Cursor sieht man jedoch nicht auf dem Bildschirm, da die Ausgabe von Programm gesteuert wird. Möchte man also die Position, an der die Ausgabe des *PRINT*-Befehls erfolgen soll, selbst bestimmen, so setzt man vor den *PRINT*-Befehl den Befehl *POSITION*. Mit diesem Befehl gibt man, wie oben

beschrieben, die Spalte und die Zeile der Position an. Ein Beispiel soll hier Klarheit schaffen. Der Text *'HALLO'* soll in der Mitte des Bildschirms ausgegeben werden. Dazu werden die Befehle *POSITION* und anschließend *PRINT* benötigt. Die Befehlsfolge sieht dann so aus:

POSITION 18,13:PRINT "HALLO"

Erinnern Sie sich: In einer Befehlszeile können mehrere Befehle untergebracht werden, wenn diese durch einen Doppelpunkt getrennt werden.

Doch nun wieder zu unserem Programm.

UNTERPROGRAMME

Im weiteren Verlauf der Adressenverwaltung werden wir die BASIC-Zeilen zur Ausgabe des Programmkopfes noch öfter benötigen. Damit wir nicht immer wieder diese Zeilen eingeben müssen, was das Programm auch unnötig verlängern würde, machen wir daraus ein Unterprogramm (auch Routine genannt). Doch wie wird ein Unterprogramm gekennzeichnet und wie führt man es aus? Zwei Fragen, die wie immer sofort geklärt werden.

Sie haben bereits im vorherigen Kapitel den Befehl *GOTO* kennengelernt, der es ermöglicht, das Programm in irgendeiner Zeile fortzusetzen. Da nach Ablauf eines Unterprogrammes jedoch wieder zurück verzweigt werden muß, und der Rechner nicht wissen kann, von wo mit *GOTO* aufgerufen wurde, ist der Befehl *GOTO* hier fehl am Platze. Zum Aufruf von Routinen gibt es einen ähnlichen Befehl, der nun zunächst beschrieben wird.

```
Problem:    Aufruf von Unterprogrammen (Routinen)

Befehl:     GOSUB zn

Parameter:  zn - Zeilennummer, ab der die Routine be-
                 ginnt

Beispiel:   GOSUB 100
            ruft ein Unterprogramm bei Zeile 100 auf

Bemerkung:  - Die mit GOSUB aufgerufenen Unterpro-
              gramme müssen mit dem Befehl RETURN ab-
              geschlossen werden.
            - Stößt das Programm während des Ablaufs
              auf ein RETURN, ohne daß ein GOSUB-Be-
              fehl ausgeführt wurde, so führt dies zur
              Fehlermeldung "ERROR-16"
            - Ist die angegebene Zeilennummer im Pro-
              gramm nicht vorhanden, so folgt die Mel-
              dung "ERROR-12"
```

Dies sieht eigentlich recht einfach aus. Ist es auch, wenn man nur einige Male damit gearbeitet hat. Wir machen nun die Zeilen 100-170 zu einem Unterprogramm, das von beliebiger Stelle im späteren Programm aufgerufen werden kann. Wie Sie der Befehlsbeschreibung entnommen haben, müssen wir dazu dieses Unterprogramm mit dem Befehl *RETURN* abschließen. Dazu hängen wir eine Zeile an:

```
180 RETURN
```

Nun dürfen wir dieses Programm jedoch nicht mehr starten. Machen Sie es doch, so erscheint die Fehlermeldung "*ERROR-16 AT LINE 180*" (Fehler Nummer 16 in Zeile 180). Dies ist auch klar, denn wir steigen mit *RUN* einfach in dieses Unterprogramm ein, das nur mit *GOSUB* aufgerufen werden darf. Wie machen wir nun weiter?
Beachten Sie zunächst, wie die Zeilennummern des zukünftigen Programms organisiert werden.

0-99: *Hier sollen alle Vorabinformationen für das Programm untergebracht werden. Z.B. Einrichten von Tabellen und Variablen*

100-999: *In diesen Zeilen werden alle Routinen untergebracht.*

1000-: *Ab der Zeile 1000 beginnt das eigentliche Programm, das Hauptprogramm.*

Wir müssen also das Unterprogramm überspringen, um ins zu gelangen. Dazu geben wir die Zeile 99 ein:

```
99 GOTO 1000
```

In Zeile 1000 beginnt nun das Hauptprogramm. Dies kennzeichnen wir zunächst mit *REM*-Zeilen und setzen anschließend den Befehl GOSUB 100 ein:

```
1000 REM ====================
1010 REM    HAUPTPROGRAMM
1020 REM ====================
1030 GOSUB 100
```

Nun ist das Programm wieder lauffähig. Es wird zuerst das Unterprogramm zum Anzeigen des Programmkopfes ausgeführt.

DAS MENÜ

Schauen wir zunächt, welche Fortschritte unser Programm gemacht hat:

```
99 GOTO 1000
100 REM ===================
110 REM    PROGRAMMKOPF
120 REM ===================
130 PRINT CHR$(125)
140 FOR I=1 TO 38:PRINT "=";:NEXT I
150 POSITION 2,12
```

```
160 PRINT "ADRESSENVERALTUNG"
170 FOR I=1 TO 38:PRINT"=";:NEXT I
180 RETURN
1000 REM ====================
1010 REM    HAUPTPROGRAMM
1020 REM ====================
1030 GOSUB 100
```

Das Programm meldet sich nun nach dem Starten mit seinem Namen. Doch dies ist sicherlich nicht alles, sondern lediglich ein kleiner Anfang. Da wir mit diesem Programm mehrere Möglichkeiten haben möchten, die Adressen zu bearbeiten, müssen wir diese auch anzeigen. Überlegen wir, was das Programm alles können soll. Zunächst soll es die Daten auf ein externes Speichermedium speichern und von dort wieder laden können. Wir benötigen dazu die beiden folgenden Auswahlmöglichkeiten:

-1- DATEI LADEN
-2- DATEI SPEICHERN

Weiterhin müssen wir natürlich Adressen eingeben. Dazu benötigen wir die Funktion 3:

-3- ADRESSEN EINGEBEN

Die eingegebenen Daten müssen auch zu ändern sein. Der folgende Punkt 4 soll dies ermöglichen:

-4- ADRESSEN AENDERN

Wenn wir vom Onkel Karl ab sofort nichts mehr wissen möchten, sollten wir ihn aus der Adreßdatei nehmen. Wir benötigen also eine Funktion "löschen":

-5- ADRESSEN LOESCHEN

Nun kommt das wichtigste: Alle Daten nützen uns wenig, wenn wir diese nicht ausgeben, also abrufen können. Mit folgendem Teilprogramm sollen Adressen ausgegeben werden:

-6- ADRESSEN AUSGEBEN

Kein Programm sollte mit dem Netzschalter oder der RESET-Taste des Rechners beendet werden. Wir schaffen deshalb mit dem letzten Punkt den Programmausgang:

-7- PROGRAMM BEENDEN

Nachdem wir jetzt über alle Programmfunktionen im klaren sind, erweitern wir unser Programm so, daß diese Funktionen als Menü ausgegeben werden. Die folgenden Anweisungen werden dazu hinzugefügt:

```
1040 POSITION 1,7
1050 PRINT"     PROGRAMMFUNKTIONEN:"
1060 PRINT"     -------------------"
1070 PRINT
1080 PRINT"       -1- DATEI LADEN"
1090 PRINT"       -2- DATEI SPEICHERN"
1100 PRINT"       -3- ADRESSEN EINGEBEN"
1110 PRINT"       -4- ADRESSEN AENDERN"
1120 PRINT"       -5- ADRESSEN LOESCHEN"
1130 PRINT"       -6- ADRESSEN AUSGEBEN"
1140 PRINT"       -7- PROGRAMM BEENDEN"
```

Wenn Sie das Programm nun noch einmal starten, werden Sie erkennen, daß es langsam einen professionellen Charakter bekommt. Nun muß der Anwender des Programms entscheiden, welche Programmfunktionen er benötigt. Wir lesen dazu eine numerische Variable ein, die den Wert 1 bis 7 entsprechend der ausgewählten Funktion enthalten soll. Vorher wird noch ein *PRINT* mit dem Inhalt "*AUSWAHL*" ausgegeben.

```
1150 POSITION 10,18:PRINT"AUSWAHL: 0 ";
1160 POSITION 18,18:INPUT FUNKTION
```

Wenn Sie das Programm wieder starten, sehen Sie, daß das Programm eine Eingabe erwartet. Die Null ist hier vorgegeben, da das Programm mit einer Fehlermeldung aussteigt, wenn beim Einlesen einer numerischen Variablen nichts eingegeben wird. Dies erreichen wir, indem der *INPUT*-Befehl mit dem POSITION-Befehl auf die Null aufgesetzt wird.

DIE ABFRAGE MIT IF

Was aber, wenn der Anwender eine ungültige Zahl eingibt, z.B. 9? Dies sollte vom Programm unterbunden werden. Die Frage ist nur wie? Wir müssen feststellen, welcher Wert eingegeben wurde, also die Variable *FUNKTION* auf einen ungültigen Wert abfragen. Die Anweisung dazu wird nun beschrieben:

```
Problem:    Abrage auf Bedingungen

Befehl:     IF bedingung THEN anweisung

Parameter:  bedingung - Vergleichoperation
                        (z.B. A=10 oder B=12)
            anweisung - Befehle, die ausgeführt wer-
                        den, wenn die Bedingung wahr
                        ist

Beispiel:   IF F=0 THEN GOTO 1000
            Wenn die Variable F den Wert 0 hat, wird
            zur Zeile 1000 verzweigt, ansonsten wird
            in der Zeile danach das Programm fortge-
            setzt.

Bemerkung:  Folgt dem THEN ein GOTO, so kann dieses
            GOTO ignoriert werden. Das letzte Bei-
            spiel könnte auch so eingegeben werden:
            IF F=0 THEN 1000
```

Eine Bedingung kann nicht nur den Vergleichsoperator '=' beinhalten, sondern auch die Operatoren '>' (größer als) und '<' (kleiner als). Auch Kombinationen der drei sind erlaubt. Alles in allem können folgende Vergleichoperatoren eingesetzt werden:

Symbol	**Bedeutung**
=	*gleich*
>	*größer als*
<	*kleiner als*
>=, =>	*größer als oder gleich*
<=, =<	*kleiner als oder gleich*
<>, ><	*ungleich*

Der *IF*-Befehl prüft also erst, ob die Bedingung wahr ist. Ist dies der Fall, so werden die Anweisungen hinter dem *THEN* ausgeführt. Trifft die Bedingung nicht zu, so wird das Programm hinter der *IF*-Zeile fortgesetzt.

Doch dies ist nicht alles, was eine *IF*-Abfrage zu leisten vermag. Es können auch mehrere Bedingungen logisch miteinander verknüpft werden. Dies hört sich komplizierter an, als es eigentlich ist. Nehmen wir hier ein Beispiel zu Hilfe:

IF ANTW=1 OR ANTW>12 THEN 1000

Zwei Vergleiche, die miteinander verknüpft wurden. *OR* heißt *ODER* und somit kann der Befehl wie folgt interpretiert werden: Wenn *ANTW* gleich 1 oder *ANTW* größer als 12, dann verzweige nach 1000. Zum Sprung nach Zeile 1000 muß also entweder *ANTW=1* oder *ANTW>12* oder beides zutreffen. Nur wenn beide Bedingungen verneint werden, wird das Programm hinter der *IF*-Zeile fortgesetzt.

Wahrheitstabelle OR

Bedingung 1	Bedingung 2	Resultat
nein	nein	nein
ja	nein	ja
nein	ja	ja
ja	ja	ja

Die *OR*-Verknüpfung ist eine der möglichen Verknüpfungen. Ein weiteres Beispiel:

IF ANTW=1 AND TEST>12 THEN 1000

Dies ist eine *UND*-Verknüpfung, die nur dann erfüllt ist, wenn *ANTW* gleich 1 und *TEST* größer als 12 ist.

Auch hier die Wahrheitstabelle:

Wahrheitstabelle AND

Bedingung 1	Bedingung 2	Resultat
nein	nein	nein
ja	nein	nein
nein	ja	nein
ja	ja	ja

Kommen wir zu unserem Problem zurück. Wir wollten abfragen, ob der Anwender eine Zahl eingegeben hat, die nicht in dem Bereich von 1-7 liegt. Wie muß die entsprechende *IF*-Anweisung codiert werden? Schauen Sie sich die Lösung an:

IF FUNKTION=0 OR FUNKTION>7 THEN ...

Ja, was dann? Geben wir einfach eine Fehlermeldung in der letzten Bildschirmzeile aus und gehen zurück zur Zeile 1000. Zur Ausgabe der Fehlermeldung benötigen wir ein weiteres Unterprogramm, das die Art des Fehlers einem String entnimmt und diesen in Zeile 23 ausgibt. Sehen wir uns dieses Programm zunächst einmal an:

```
200 REM ===================
210 REM    FEHLERMELDUNG
220 REM ===================
230 POSITION 1,23
240 PRINT FEHLER$;
250 PRINT CHR$(253);CHR$(253);:REM 2 MAL PIEP
260 POSITION 1,23
270 PRINT CHR$(156);:REM ZEILE LOESCHEN
280 RETURN
```

Zunächst wird in Zeile 230 der Cursor in die letzte Zeile positioniert. Die Zeile 240 gibt den String *FEHLER$*, der vom aufrufenden Programm gesetzt wurde, aus. Wie Sie sicher noch wissen, muß jede Stringvariable am Programmanfang eingerichtet werden. Dafür nehmen wir uns den Bereich von Zeile 1 bis Zeile 99. Tragen wir hier nun die erste Variable ein:

```
1 REM --------------------
2 REM    STRINGVARIABLEN
3 REM --------------------
4 DIM FEHLER$(38)
```

Hier tragen wir auch die im weiteren Verlauf noch anfallenden Stringvariablen ein.

Bevor die Fehlerroutine vom Hauptprogramm aufgerufen wird, muß in der Stringvariablen *FEHLER$* die auszugebende Fehlermeldung gespeichert werden.

ASCII-CODE

Nun zu der Funktion in Zeile 250, die wir bereits zum Löschen des Bildschirms eingesetzt haben. Doch gehen wir näher auf diese Funktion ein. Jedes Zeichen wird im Rechner codiert. Es erhält einen Wert zwischen 0 und 255. Diese Codierung ist genormt nach der amerikanischen ASCII-Norm. Die ATARI-Zeichen weichen geringfügig von diesem Code ab. Doch nicht nur Zeichen werden mitcodiert. Auch spezielle Funktionen, wie in unserem Programm das Ausgeben eines kurzen Tones, können mit einem ASCII-Code ausgegeben werden. Zum Codieren der Zeichen nach dem ASCII-Code gibt es zwei Funktionen, die zunächst beschrieben werden.

```
Problem:    Umwandlung eines ASCII-Codes in ein
            Zeichen

Funktion:   CHR$(n)

Parameter:  n - ASCII-Code (0-255)

Beispiel:   FOR I=65 TO 90: PRINT CHR$(I);:NEXT I
            zeigt alle mit dem ATARI darstellbaren
            Großbuchstaben


Problem:    Umwandlung eines Zeichens in den ASCII-
            Code

Funktion:   ASC("z")

Parameter:  "z" - ein Zeichen, oder eine Stringvari-
                  able, die ein Zeichen enthält

Beispiel:   PRINT ASC("A")
            zeigt den ASCII-Code des Buchstabens "A"
            (65) auf dem Bildschirm an.
```

```
Bemerkung: - Ist die Stringvariable leer, so wird
             der ASCII-Code des Zeichens ausgegeben,
             das sich in der Speicherstelle befindet,
             an der der Rechner den String angelegt
             hat (ein Zufallswert).
           - Ist die Stringvariable länger als ein
             Zeichen, so wird der ASCII-Code des
             ersten Zeichens ermittelt.
```

Dies ist eine Menge an Informationen, die erst einmal verdaut werden muß. Wie gesagt, arbeitet der Rechner intern nur mit den ASCII-Codes der Zeichen. Demnach versteht er auch, wenn Sie diesen ASCII-Code und nicht das Zeichen selbst eingeben.
Wenn Sie wissen möchten, welchen ASCII-Code ein bestimmtes Zeichen hat, so ermitteln Sie dies mit der Funktion *ASC("z")*. Sie geben dazu im Direktmodus nur den Befehl PRINT *ASC("z")* - für *z* setzen Sie das gewünschte Zeichen - ein. Der ATARI antwortet dann prompt mit dem entsprechenden Code.

In Zeile 250 wird also mit dem Befehl '*PRINT CHR$(253)*' ein Ton erzeugt, der akustisch auf einen Fehler hinweisen soll. Einen weiteren Code finden Sie in Zeile 270. Hier wird mit '*PRINT CHR$(156)*' die aktuelle Zeile gelöscht. Im Anhang finden Sie eine ausführliche Liste sämtlicher Codes.

Doch weiter in unserer Fehlerroutine. Dem akustischen Signal folgt die Positionierung auf die Fehlerzeile, die anschließend wieder gelöscht wird. In der Zeile 280 erfolgt schließlich der Rücksprung zum Hauptprogramm.

Wir setzen nun unser Hauptprogramm mit der Abfrage der Menüauswahl fort.

```
1170 IF FUNKTION=0 OR FUNKTION>7 THEN FEHLER$="UNGUELTIGER WERT"
:GOSUB 200:GOTO 1160
```

Geben Sie diese Zeile nun ein. Beachten Sie, daß die Aufteilung der zwei Textzeilen nicht der Aufteilung auf Ihrem Bildschirm entspricht.

Wenn die eingegebene Zahl nun 0 oder größer als 7 ist, liegt eine Fehlbedienung des Anwenders vor, auf den wir ihn in der Routine 200 hinweisen. Dazu wird zunächst die Fehlermeldung im String untergebracht, dann die Fehlerroutine aufgerufen und anschließend zum erneuten *INPUT* verzweigt.

Starten Sie nun das Programm und geben Sie einen falschen Wert ein. Das Programm stellt die Falscheingabe fest und gibt die Fehlermeldung aus.

BERECHNETES GOTO

Wenn ein gültiger Wert eingegeben wurde, müssen wir die nötigen Schritte einleiten. Das heißt, es muß zum entsprechenden Programmteil verzweigt werden.

Um abhängig von der Variablen '*FUNKTION*' im Programm zu verzweigen, gibt es einen Befehl, der dies auf leichteste Art und Weise ermöglicht. Schauen Sie sich die Beschreibung dieses Befehls an:

```
Problem:    Berechneter Sprung

Befehl:     ON var GOTO z1,z2,z3,...

Parameter:  var            - numerische Variable
            z1,z2,z3,...   - Zeilennummer, zu denen ab-
                             hängig vom Inhalt der Va-
                             riablen 'var' gesprungen
                             werden soll
```

```
Beispiel: ON F GOTO 100,200,300,400
          Ist F=1, dann wird zur Zeile 100 verzweigt,
          ist F=2, dann wird zur Zeile 200 verzweigt,
          usw. Ist F größer als 4, so wird nicht ver-
          zweigt, sondern in der nächsten Zeile wei-
          tergemacht.

Bemerkung: Ist F keine ganze Zahl (z.B. 3.45), so
           wird immer die Zahl vor dem Komma be-
           nutzt.
```

Ein idealer Befehl also für unseren Zweck. Wir bestimmen zuerst, bei welchen Zeilennummern die Programmteile begonnen werden sollen.

'FUNKTION'	Zeile	Programmteil
1	5000	DATEI LADEN
2	10000	DATEI SPEICHERN
3	15000	ADRESSEN EINGEBEN
4	20000	ADRESSEN AENDERN
5	25000	ADRESSEN LOESCHEN
6	30000	ADRESSEN AUSGEBEN
7	31000	PROGRAMM BEENDEN

Dementsprechend sieht unser nächster Befehl wie folgt aus:

```
1180 ON FUNKTION GOTO 5000,10000,15000,20000,25000,30000,31000
```

Wir werden nun einen Programmteil nach dem anderen fertigstellen. Sinngemäß fangen wir mit der Funktion 3, *ADRESSEN EINGEBEN* an. Doch zunächt müssen wir noch die Tabelle zur Aufnahme der Adressen einrichten.

Wir wollen Platz für 100 Adressen reservieren. Das dürfte in der Regel vollkommen ausreichen. Wer hat schon mehr als 100 Bekannte? Für den kommerziellen Einsatz ist das Programm nur beschränkt einsatzfähig. Eine Firma, die ihre 2000 Kunden verwalten möchte, sollte sich nach einem anderen geeigneten Programm umsehen.

Erinnern Sie sich, wie wir die Adressentabelle geplant haben? Hier nochmal die Tabellenstruktur:

Adressentabelle TABELLE$

Feldnr.	Bezeichnung	Länge	Position im Satz
1	ANREDE	20	1,20
2	VORNAME	20	21,40
3	NAME	20	41,60
4	STRASSE	20	61,80
5	PLZ/ORT	20	81,100
6	TELEFON	20	101,120
7	BEMERKUNG	20	121,140

Gesamtlänge pro Satz	140 Zeichen
Anzahl Sätze	100
Tabellenlänge	14000 Zeichen

Zunächst reservieren wir Speicherplatz für die Tabelle, um diese dann mit Leerzeichen zu füllen:

```
5 DIM TABELLE$(14000)
6 TABELLE$(1)=" ":TABELLE$(14000)=" "
7 TABELLE$(2)=TABELLE$
```

Da die Adressentabelle nun reserviert ist, können wir mit unserem Hauptprogramm fortfahren:

Lassen Sie uns vor den einzelnen Programmteilen eine Routine schreiben, die für jedes Programm einen schönen Kopf ausgibt. Diese Routine soll vorher auch den Programmkopf ausgeben. Jeder Programmteil hat einen Namen, der auch im Menü angegeben wurde. Diese Namen legen wir in einer Tabelle ab. Mittels der Variablen *'FUNKTION'* kann aus der Tabelle dann der Name des Programmteils ermittelt und ausgegeben werden. In der Übersicht auf der vorletzten Seite sehen Sie, daß z.B. der Programmteil *'ADRESSEN EINGEBEN'* die Nummer 3 hat. Diese Nummer ist dann der "Schlüssel" für den Tabellenplatz. Der Tabelle geben wir den Namen *'FUNKTION$'*. Da der Name des längsten Teilprogramms 17 Zeichen umfasst, richten wir eine Tabelle mit 17 mal 7, also 119 Zeichen ein und füllen diese diesmal nicht mit Leerzeichen auf, da die Namen, die in dieser Tabelle gespeichert werden, alle 17 Zeichen ausnutzen.

```
8 DIM FUNKTION$(119)
```

Diese Tabelle wollen wir anschließend mit den Namen der Programmteile füllen:

```
9 FUNKTION$(1,17)   ="  DATEI LADEN    "
10 FUNKTION$(18,34)  =" DATEI SPEICHERN "
11 FUNKTION$(35,51)  ="ADRESSEN EINGEBEN"
12 FUNKTION$(52,68)  ="ADRESSEN AENDERN "
13 FUNKTION$(69,85)  ="ADRESSEN LOESCHEN"
14 FUNKTION$(86,102) ="ADRESSEN AUSGEBEN"
15 FUNKTION$(103,119)="PROGRAMM BEENDEN "
```

So kann mit der Auswahl FUNKTION auf den Namen zugegriffen werden. Die Formel dazu lautet:

*FUNKTION$ (FUNKTION*17-16,FUNKTION*17)*

Beachten Sie dabei, daß alle Strings die gleiche Länge erhalten (zweites Anführungszeichen untereinander). In dem Unterprogramm ab Zeile 300 wollen wir nun den Programmkopf und anschließend die vom Anwender ausgewählte Funktion anzeigen:

```
300 REM ====================
310 REM KOEPFE PROGRAMMTEILE
320 REM ====================
330 GOSUB 100
340 POSITION 10,5
350 FOR I=1 TO 19:PRINT"*";:NEXT I
360 POSITION 10,6
370 PRINT "*";FUNKTION$(FUNKTION*17-16,FUNKTION*17);"*"
380 POSITION 10,7
390 FOR I=1 TO 19:PRINT"*";:NEXT I
395 RETURN
```

Hier sehen Sie, daß auch ein Unterprogramm ein anderes Unterprogramm aufrufen kann. Es wird zunächst der allgemeine Programmkopf ausgegeben. Danach wird die erste Zeile des Kästchens, das den Namen der Funktion umhüllen soll, erzeugt. Die nächste Zeile sieht komplizierter aus, als sie eigentlich ist. Zunächst wird der Cursor auf die 10. Spalte der 6. Zeile positioniert. Dort wird dann ein Sternchen, dann der Name der Funktion ausgegeben. Dieser Name wird der Tabelle '*FUNKTION$*' mit der bekannten Formel entnommen. An diesen String wird dann ein weiteres Sternchens angehängt. Die Zeile 390 erzeugt die untere Zeile des Rahmens. Die letzte Zeile muß nicht mehr beschrieben werden.

Doch wann wird diese Routine aufgerufen. Logischerweise kann sie erst ausgeführt werden, wenn der Anwender bereits eine Funktion ausgewählt hat, deren Nummer in der Variablen '*FUNKTION*' enthalten ist. Also hinter der *IF*-Abfrage. Fügen wir nun die Zeile 1175 ein:

```
1175 GOSUB 300
```

Wir müssen diese Zeile nehmen, da nach der Zeile 1180 in einen anderen Teil des Programms verzweigt wird.

ADRESSEN EINGEBEN

Nun wird es ernst. Der erste anspruchsvolle Programmteil muß erstellt werden. Doch fangen wir erst einmal mit dem einfachsten an:

```
15000 REM ====================
15010 REM  ADRESSEN EINGEBEN
15020 REM ====================
```

So weit, so gut. Jetzt wird es kompliziert (nicht den Mut verlieren). Wir müssen uns irgendwie merken, wieviel Datensätze eingegeben wurden. Dies ist deshalb notwendig, damit nicht immer die gesamte Adressentabelle gespeichert und geladen werden muß, wenn nur wenige Adressen eingegeben wurden. Dazu erhöhen wir in diesem Programteil stets einen Zeiger um eins, der die Anzahl der eingegebenen Adressen erhält. Nennen wir ihn sinngemäß '*ZEIGER*'. Die nächste Zeile soll diesen Zeiger um eins erhöhen:

```
15030 ZEIGER=ZEIGER+1
```

Wenn z.B. '*ZEIGER*' ungleich 0 ist, so wurden vorher bereits Daten geladen oder eingegeben, die dann hier erweitert werden.

Nun müßten wir 7 *INPUT*-Zeilen eingeben, um die 7 Datenfelder einer Adresse einzulesen. Doch dies geht einfacher. Dazu legen wir eine weitere Tabelle an, die die sieben Feldbezeichnungen enthält. Die Feldlänge soll 10 Zeichen betragen. Da unsere Adresse aus 7 Feldern besteht, muß die Tabelle 70 Zeichen umfassen. Diese Tabelle reservieren wir und füllen sie anschließend mit den Feldbezeichnungen:

```
16 DIM FELD$(70)
17 FELD$(1,10) ="ANREDE    "
18 FELD$(11,20) ="VORNAME   "
19 FELD$(21,30) ="NAME      "
20 FELD$(31,40) ="STRASSE   "
21 FELD$(41,50) ="PLZ/ORT   "
22 FELD$(51,60) ="TELEFON   "
23 FELD$(61,70) ="BEMERKUNG "
```

Mit der Feldnummer (1 bis 7) kann nun mit folgender Formel die Feldbezeichnung ermittelt werden:

*FELD$(INDEX*10-9,INDEX*10)*

Diese Feldnamen können im gesamten Programmablauf eingesetzt werden. Wozu wir sie jetzt benötigen, werden Sie im folgenden Verlauf des Programteils feststellen.

```
15040 PRINT
15050 FOR INDEX=1 TO 7
15055 GOSUB 600
15060 PRINT FELD$(INDEX*10-9,INDEX*10)
15065 INPUT ZW$
15070 TABELLE$(VON,BIS)=ZW$
15080 NEXT INDEX
```

Hier werden die sieben Felder einer Adresse mit einer Schleife eingelesen. In der Schleife wird zunächst die Routine ab Zeile 600 aufgerufen. Hier soll mit Hilfe der Satznummer (*ZEIGER*) und der Feldnummer (*INDEX*) der Bereich ermittelt werden, den das Feld in der Tabelle belegt. Die Anfangsposition soll in die Variable '*VON*' und die Endposition in '*BIS*' gespeichert werden. Sehen wir uns diese Routine an und geben sie auch gleich ein:

```
600 REM ===================
610 REM  ADRESSEN-POSITION
620 REM ===================
630 VON=(ZEIGER*140-139)+(INDEX*20-20)
640 BIS=VON+19
650 RETURN
```

Der Algorithmus in Zeile 630 ist bereits beschrieben worden. Nach Aufruf dieser Routine steht also die Feldposition in '*VON*' und '*BIS*'.

Zurück zum Programmausschnitt 15040-15080. Nach Ermittlung der Feldposition wird in der Zeile 15060 die Feldbezeichnung ausgegeben. Unmittelbar dahinter wird der Inhalt dieses Feldes mit *INPUT* in den Zwischenspeicher ZW$ abgelegt. Diese Variable müssen wir noch reservieren:

```
24 DIM ZW$(20)
```

Das eingelesene Feld wird dann an die entsprechende Position in der Adressentabelle gespeichert. Das Schleifenende wird in Zeile 15080 bestimmt.

Diese Programmschleife liest eine komplette Adresse ein und speichert sie in die Tabelle.

Starten Sie das Programm, nachdem Sie diese Zeilen eingegeben haben und wählen die Funktion 3 aus. Was nun geschieht, bewirkt allein diese ausgetüftelte Schleife. Ein gesamter Adressensatz wird mit wenigen Befehlen eingelesen.

Nachdem wir die Felder eingelesen haben, müssen wir dem Anwender die Möglichkeit geben, die Eingabe zu wiederholen. Er kann sich irgendwo verschrieben haben und muß dann korrigieren.

```
15090 POSITION 2,22
15100 PRINT "DATEN RICHTIG EINGEGEBEN (J/N)";
15110 INPUT ANTWORT$
15120 IF ANTWORT$="J"THEN 15150
15130 IF ANTWORT$="N" THEN ZEIGER=ZEIGER-1:GOSUB 300:GOTO 15000
15140 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 15090
```

Richten Sie bitte gleich die neue Stringvariable ANTWORT$ ein:

```
25 DIM ANTWORT$(1)
```

Wenn die Daten richtig eingegeben wurden, so wird das Programm fortgesetzt. Sind die Daten nicht richtig eingegeben worden, so wird der Programmteil nochmals gestartet. Vorher wird jedoch der Satzzähler um eins vermindert, da sonst in Zeile 15030 ein Datensatz weiter geschaltet wird. Wir wollen den letzten Satz jedoch nochmal eingeben. Wird keine richtige Antwort gegeben, so wird eine Fehlermeldung ausgegeben und anschließend neu eingelesen.

Das Programm wird nun mit einer weiteren Frage fortgesetzt:

```
15150 POSITION 2,23
15160 PRINT "WEITERE EINGABEN (J/N)";
15170 INPUT ANTWORT$
15180 IF ANTORT$="J" THEN GOSUB 300:GOTO 15000
15190 IF ANTWORT$="N" THEN 1000
15200 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:
GOTO 15150
```

Hier muß sich der Anwender entscheiden, ob er weitere Adressen eingeben möchte oder nicht. Wenn ja, dann wird der Programmteil erneut gestartet. Wenn nein, so wird zurück ins Menü verzweigt. Eine falsche Eingabe führt wieder zur Fehlermeldung und erneuten Abfrage.

Damit haben wir den ersten Programmteil abgeschlossen. Man kann nun beliebig viele Daten eingeben, aber noch nicht viel damit anfangen. Zum Schluß folgt nochmals der gesamte Programmteil "*ADRESSEN EINGEBEN*!", den Sie mit Ihrem nun vergleichen können:

```
15000 REM ===================
15010 REM ADRESSEN EINGEBEN
15020 REM ===================
15030 ZEIGER=ZEIGER+1
15040 PRINT
15050 FOR INDEX=1 TO 7
15055 GOSUB 600
15060 PRINT FELD$(INDEX*10-9,INDEX*10)
15065 INPUT ZW$
15070 TABELLE$(VON,BIS)=ZW$
15080 NEXT INDEX
15090 POSITION 2,22
15100 PRINT "DATEN RICHTIG EINGEGEBEN (J/N)";
15110 INPUT ANTWORT$
15120 IF ANTWORT$="J" THEN 15150
15130 IF ANTWORT$="N" THEN ZEIGER=ZEIGER-1:GOSUB 300:GOTO 15000
15140 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 15090
15150 POSITION 2,23
15160 PRINT "WEITERE EINGABEN (J/N)";
15170 INPUT ANTWORT$
15180 IF ANTWORT$="J" THEN GOSUB 300:GOTO 15000
15190 IF ANTWORT$="N" THEN 1000
15200 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 15150
```

ADRESSEN ÄNDERN

Nun möchten wir dem Anwender unseres Programms die Möglichkeit geben, einzelne Felder eines bestimmten Datensatzes zu ändern. Wir benutzen dazu drei Tasten. Mit der Taste '+' soll vorwärts und mit der Taste '-' rückwärts "geblättert" werden. Soll eine Adresse geändert werden, so wird die Taste '*' gedrückt. Mit der Taste 'M' wird zurück ins Menü verzweigt, die Änderung also abgeschlossen.

Das hört sich alles gut an, muß aber erst programmiert werden. Doch fangen wir zunächst mit den ersten Zeilen an:

```
20000 REM ===================
20010 REM  ADRESSEN AENDERN
20020 REM ===================
```

Diese Zeilen bedürfen keines Kommentars. Jetzt aber wird es kritisch. Wie gehen wir dieses Problem an? Wir wählen uns zunächst eine Variable aus, die den ersten Adressensatz darstellen soll. Dies ist die Varibale '*ZAEHLER*', die den Anfangswert 1 erhält.

```
20030 ZAEHLER=1:Z=ZEIGER
```

Zusätzlich wurde der Zeiger auf den letzten Datensatz in die Variable '*Z*' gerettet. Da die Routine zur Berechnung der Adressenposition in der Tabelle (ab 600) mit der Variable '*ZEIGER*' arbeitet. Diese Routine wollen wir hier auch benutzen.

Nun können wir die erste Adresse in der Tabelle ausgeben. Sie besteht bekanntlich aus 7 Feldern, deren Namen in einer weiteren Tabelle gespeichert sind. Wir codieren wieder eine Schleife, die die Nummer des Feldes, den Namen und den Inhalt ausgibt.

```
20040 POSITION 2,10
20050 FOR INDEX=1 TO 7
20055 ZEIGER=ZAEHLER:GOSUB 600
20060 PRINT INDEX;"-";FELD$(INDEX*10-9,INDEX*10);
20065 PRINT TABELLE$(VON,BIS)
20070 NEXT INDEX
```

Es werden also 7 Zeilen ausgegeben, die jeweils die Nummer des Feldes, dessen Bezeichnung und den Inhalt enthält. Der Inhalt des Feldes ergibt sich aus dem Index '*ZAEHLER*', der bekanntlich auf den Datensatz zeigt, und dem eigentlichen Feldindex ('*INDEX*'). Da das Unterprogramm ab Zeile 600 den Satzindex in der Variablen *ZEIGER* verlangt, speichern wir '*ZAEHLER*' in diese Variable, die wir vorher gerettet haben. Nach Beendigung dieses Programmteils muß der *ZEIGER* selbstverständlich wieder auf den alten Wert (Z) gebracht werden.

In der Zeile 20060 wird zuerst die Variable *INDEX* ausgegeben, die die Feldnummer enthält. Der anschließende Bindestrich trennt die Feldnummer optisch von der folgenden Feldbezeichnung. In der nächsten Zeile wird dann der eigentliche Inhalt des Feldes ausgegeben.

Nachdem die erste Adresse ausgegeben wurde, muß der Anwender weiter entscheiden. Er hat vier Möglichkeiten:

Taste '+'	*vorwärts blättern (nächste Adresse)*
Taste '-'	*rückwärts blätten (letzte Adresse)*
Taste ''*	*angezeigte Adresse ändern*
Taste 'M'	*zurück ins Menü*

Nun müssen wir das Kommando einlesen und anschließend reagieren. Wir benutzen wieder den Befehl *INPUT*:

```
20075 POSITION 5,18
20080 PRINT "KOMMANDO (+ - * M)";
20085 INPUT ANTWORT$
```

Wir müssen nun abfragen, welche Kommandotaste gedrückt wurde und entsprechend reagieren. Zunächst fragen wir die Taste 'M' ab:

```
20090 IF ANTWORT$="M" THEN ZEIGER=Z:GOTO 1000
```

Wir verzweigen also zum Menü, wenn diese Taste gedrückt wurde. Doch zuvor wird die Variable *'ZEIGER'* wieder rekonstruiert. Wird die Taste '+' gedrückt, so soll der nächste Datensatz ausgegeben werden. Wir erhöhen dazu den Index *'ZAEHLER'* um 1, aber nur wenn *'ZAEHLER'* nicht schon den letzten Satz erreicht hat (*ZAEHLER=Z*). Die folgende Zeile erfüllt diese Aufgabe:

```
20100 IF ANTWORT$="+" AND ZAEHLER<Z THEN ZAEHLER=ZAEHLER+1:GOTO 20040
```

Nur wenn die Taste '+' gedrückt wurde **und** die letzte Adresse noch nicht erreicht ist, wird der Zähler auf die nächste Adresse gesetzt und diese dann ausgegeben.

Bei der Taste '-' ist es ähnlich, nur muß '*ZAEHLER*' um 1 vermindert werden, wenn '*ZAEHLER*' nicht schon 1 ist, weil dann die erste Adresse vorliegt.

```
20110 IF ANTWORT$="-" AND ZAEHLER>1 THEN ZAEHLER=ZAEHLER-1:GOTO 20040
```

Bei Zeile 20040 wird die Adresse mit dem Index '*ZAEHLER*' ausgegeben. Dieser Zähler wurde hier um eins vermindert.

Nun müssen wir noch abfragen, ob die Taste '*' gedrückt wurde. Mit dieser Taste wird die Änderung eingeleitet.

```
20120 IF ANTWORT$="*" THEN 20140
20130 FEHLER$="EINGABEFEHLER!":GOSUB 200:GOTO 20075
```

Die Zeile 20130 wird erreicht, wenn die Tasten '+' oder '-' die Datei über- oder unterschreiten wollten oder wenn kein gültiges Kommando eingegeben wurde. Dann werden die Tasten einfach ignoriert, eine Fehlermeldung ausgegeben und zur Zeile 20075 verzweigt. Ab dieser Zeile wird erneut ein Kommando verlangt. Nun fehlt nur noch die eigentliche Änderung des Datensatzes ab Zeile 20140. Dort soll zuerst nach der Nummer und anschließend nach dem neuen Inhalt des zu ändernden Feldes gefragt werden.

```
20140 POSITION 5,19
20145 PRINT "FELDNUMMER (1-7): 0 ";
20150 POSITION 22,19:INPUT INDEX
20155 IF INDEX>0 AND INDEX<8 THEN 20165
20160 FEHLER$="NUMMER 1-7 EINGEBEN!":GOSUB 200:
GOTO 20150
20165 ZEIGER=ZAEHLER:GOSUB 600
20170 PRINT "NEUER INHALT: ";
20180 INPUT ZW$
20185 TABELLE$(VON,BIS)=ZW$
20190 GOSUB 300:GOTO 20040
```

Dies war schon der Rest unseres Programmteils. Zunächst wird die Nummer des zu ändernden Feldes in *'INDEX'* eingelesen. Danach wird getestet, ob *'INDEX'* einen gültigen Wert enthält. Ist dies der Fall, so wird ab Zeile 20170 der neue Inhalt dieses Feldes eingelesen und der neu entstandene Adressensatz ab der Zeile 20040 ausgegeben. Bei Eingabe einer ungültigen Zahl wird in Zeile 20165 eine Fehlermeldung ausgegeben. Dieser Programmteil kann nur über die Taste 'M' verlassen werden. Nun folgt auch hier die Auflistung des kompletten Teils "*ADRESSEN AENDERN*":

```
20000 REM ====================
20010 REM   ADRESSEN AENDERN
20020 REM ====================
20030 ZAEHLER=1:Z=ZEIGER
20040 POSITION 2,10
20050 FOR INDEX=1 TO 7
20055 ZEIGER=ZAEHLER:GOSUB 600
20060 PRINT INDEX;"-";FELD$(INDEX*10-9,INDEX*10);
20065 PRINT TABELLE$(VON,BIS)
20070 NEXT INDEX
20075 POSITION 5,18
20080 PRINT "KOMMANDO (+ - * M)";
20085 INPUT ANTWORT$
20090 IF ANTWORT$="M"THEN 1000
20100 IF ANTWORT$="+" AND ZAEHLER<Z THEN ZAEHLER=ZAEHLER+1:GOTO 20040
20110 IF ANTWORT$="-" AND ZAEHLER>1 THEN ZAEHLER=ZAEHLER-1:GOTO 20040
20120 IF ANTWORT$="*" THEN 20140
20130 FEHLER$="EINGABEFEHLER!":GOSUB 200:GOTO 20075
20140 POSITION 5,19
20145 PRINT "FELDNUMMER (1-7): 0 ";
20150 POSITION 22,19:INPUT INDEX
20155 IF INDEX>0 AND INDEX<8 THEN 20165
20160 FEHLER$="NUMMER 1-7 EINGEBEN!":GOSUB 200:GOTO 20150
20165 ZEIGER=ZAEHLER:GOSUB 600
20170 PRINT "NEUER INHALT: ";
20180 INPUT ZW$
20185 TABELLE$(VON,BIS)=ZW$
20190 GOSUB 300:GOTO 20040
```

ADRESSEN LÖSCHEN

Beim Löschen der Adresse wenden wir eine andere Technik als beim Ändern an. Sie sollen ja möglichst vielseitg programmieren lernen. Wir müssen den Vor- und Zunamen der zu löschenden Adresse angeben. Sind sie nicht bekannt, so kann man sie mit dem "Blättern" in "*ADRESSEN AENDERN*" ermitteln.

Doch bevor dieser Teil des Programms beginnt, soll geprüft werden, ob überhaupt Daten im Rechner gespeichert sind. Zur Ausgabe der Fehlermeldung bauen wir eine Routine ab Zeile 500 auf. Schauen wir uns zunächst die komplette Routine an:

```
500 REM ===================
510 REM     KEINE DATEN!
520 REM ===================
530 IF ZEIGER>0 THEN RETURN
540 FEHLER$="KEINE DATEN IM RECHNER!"
550 GOSUB 200
560 POP:GOTO1000
```

Diese Routine ist zwar kurz, trotzdem sinnvoll. Zunächst wird getestet, ob Daten im Rechner vorhanden sind. Wenn ja, dann wird diese Routine verlassen. Sind keine Daten vorhanden, so wird eine Fehlermeldung ausgegeben und anschließend die Routine auf für Sie vielleicht etwas verwirrende Weise verlassen. Hier können wir die Routine nicht mit *RETURN* verlassen, da das Programm nach der Fehlermeldung mit dem Menü fortgesetzt werden soll. Ein '*GOTO 1000*' allein reicht hier auch nicht aus. Erinnern Sie sich: Beim *GOSUB*-Befehl wird die Adresse der aufrufenden Zeile in einem speziellen Speicherbereich abgelegt. Der *RETURN*-Befehl weiß dann, wohin der Rücksprung erfolgen soll. Diese Adresse wird dann wieder gelöscht. Wenn ein mit *GOSUB* aufgerufenes Unterprogramm jedoch ohne *RETURN* verlassen wird, so füllt sich mit der Zeit dieser Speicher, was dann oft eine Fehlermeldung zur Folge hat. Gleiches gilt für die *FOR...NEXT*-Schleife, die ebenfalls Daten in dem speziellen Speicher ablegt und erst nach Schleifenende wieder löscht. In der

Praxis kommt es jedoch vor, daß Schleifen der Unterprogrammaufrufe vorzeitig verlassen werden müssen. Dazu kann die Rücksprungadresse mit dem Befehl *'POP'* gelöscht werden.

VOR DEM VERLASSEN EINER SCHLEIFE OHNE 'NEXT' ODER VOR DEM ABBRUCH EINES UNTERPROGRAMMS OHNE 'RETURN' MUSS DER BEFEHL 'POP' GEGEBEN WERDEN.

Wir sollten die Überprüfung auch im Programmteil "*ADRESSEN AENDERN*" machen. Wenn keine Adressen vorhanden sind, kann man auch keine ändern! Fügen Sie dazu die folgende Zeile ein:

```
20025 GOSUB 500
```

Fangen wir nun mit unserem Programmteil *'LOESCHEN'* an. Die ersten Zeilen sind erfahrungsgemäß immer die leichtesten.

```
25000 REM ====================
25010 REM  ADRESSEN LOESCHEN
25020 REM ====================
25030 GOSUB 500
25035 Z=ZEIGER
25036 V$(1)=" ":V$(20)=" ":V$(2)=V$
25037 N$(1)=" ":N$(20)=" ":N$(2)=N$
25040 POSITION 1,10
25050 PRINT "VORNAME:      ";:INPUT V$
25060 PRINT "NAME:         ";:INPUT N$
```

Wir prüfen zunächst, ob Daten im Rechner vorhanden sind. Wenn ja, speichern wir wieder den Adressenzeiger zwischen, füllen die Suchvariablen mit Leerzeichen auf und lesen anschließend Vorname und Name der zu löschenden Adresse in diese Variablen ein. Diese beiden Eingaben speichern wir in 'V$' und 'N$'. Für diese neuen Stringvariablen muß wieder vorne im Programm Platz reserviert werden. Die folgende Zeile ist dafür zuständig:

```
26 DIM V$(20),N$(20)
```

Nun müssen wir die gesamte Adressentabelle nach diesen Namen durchsuchen. Wir setzen dazu eine FOR...NEXT-Schleife ein.

```
25070 FOR ZEIGER=1 TO Z
25080 INDEX=2:GOSUB 600
25090 IF V$<>TABELLE$(VON,VON+LEN(V$)-1)) THEN 25140
25100 INDEX=3:GOSUB 600
25110 IF N$<>TABELLE$(VON,VON+LEN(N$)-1)) THEN 25140
25120 POP:GOTO 25150
25130 NEXT ZEIGER
25140 FEHLER$="ADRESSE NICHT VORHANDEN!":GOSUB 200:GOTO 1000
```

Die Schleife soll bei 1 (Dateianfang) anfangen und bis zum letzten Datensatz (Z) laufen, da ja alle Adressen durchsucht werden sollen. In der Schleife wird wieder die Routine 600 dazu benutzt, mit Hilfe der Satznummer (*ZEIGER*) und der Feldnummer (*INDEX*) die entsprechende Position zu berechnen. Bekanntlich hat der Vorname die Feldnummer 2 und der Name die Feldnummer 3. Dann wird der Suchbegriff mit dem Tabellenplatz verglichen. Hier wird es etwas kompliziert. Wir dürfen den Suchbegriff nicht mit dem mit *'VON'* und *'BIS'* angegebenen Tabellenplatz vergleichen. Der Grund dafür ist folgender: Obwohl für die Variablen zum Einlesen des Namens 20 Zeichen reserviert wurden, ist die Variable nach dem Einlesen mit *INPUT* nur so lang, wie sie eingegeben wurde. Dadurch wird dann z.B. "*MARION*" mit "*MARION*" verglichen, was dann verneint wird. Um dies zu vermeiden, darf nicht mit der gesamten Feldlänge verglichen werden. Der Vergleich muß sich auf die Länge des Feldes beschränken, die auch dem eingelesen Namen entspricht. Eine Stringfunktion des ATARI-BASIC hilft uns hier weiter:

ERMITTELN DER STRINGLÄNGE

```
Problem:    Ermitteln der Länge eines Strings

Funktion:   LEN(str)

Parameter:  str- beliebiger Stringausdruck, z.B. A$,
                 N$(1,20)) und V$(10)

Beispiel:   PRINT LEN(TABELLE$)
            zeigt die Länge des Strings TABELLE$ auf
            dem Bildschirm an

Bemerkung:  Die mit LEN ermittelte Länge muß nicht mit
            der Länge übereinstimmen, die mit DIM re-
            serviert wurde.
```

Der Stringausdruck *'TABELLE$(VON,VON+(LEN(V$)-1)'* spricht nun also nur die Länge des Namens an, die mit *INPUT* eingegeben wurde.

Es werden sowohl der Vorname als auch der Nachname verglichen. Fällt ein Vergleich negativ aus, so wird die Schleife neu durchlaufen, also mit der nächsten Adresse verglichen. Trifft das Programm auf die Zeile hinter dem Schleifenende (*NEXT*), so ist die Adresse nicht gefunden worden. Dann wird eine Fehlermeldung ausgegeben, die Variable *ZEIGER* zurückgesetzt und zum Menü verzweigt.

Nur wenn beide Namen übereinstimmen, so wird die Schleife in der Zeile 25120 vorzeitig verlassen und zur Zeile 25150 verweigt, wo die Adresse dann gelöscht wird.

Bevor die Adresse gelöscht wird, soll sie noch einmal komplett angezeigt werden. Der Anwender des Programms muß dann entscheiden, ob die Adresse wirklich gelöscht werden soll:

```
25150 GOSUB 300:POSITION 2,10
25160 FOR INDEX=1 TO 7
25170 GOSUB 600:PRINT FELD$(INDEX*10-9,INDEX*10);
25180 NEXT INDEX
25190 PRINT TABELLE$(VON,BIS)
25200 NEXT INDEX
```

Hier wird die Adresse mit den Feldbezeichnungen, die uns in der Tabelle *FELD$* zur Verfügung stehen, ausgegeben. Jetzt soll abgefragt werden, ob die Adresse wirklich gelöscht werden soll:

```
25210 POSITION 1,18
25220 PRINT "ADRESSE LOESCHEN (J/N) ";
25230 INPUT ANTWORT$
25240 IF ANTWORT$="J" THEN 25270
25250 IF ANTWORT$="N" THEN ZEIGER=Z:GOTO 1000
25260 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 25210
```

Soll diese Adresse nicht gelöscht werden (unentschlossene Leute gibt es überall), so wird zurück ins Menü verzweigt. Soll die Adresse doch gelöscht werden, so geschieht dies ab Zeile 25270. Werfen wir zunächst einen Blick in die letzten Zeilen dieses Programmteils.

```
25270 FOR SATZ=ZEIGER TO Z-1
25280 FOR INDEX=1 TO 7
25290 ZEIGER=SATZ: GOSUB 600
25300 TABELLE$(VON,BIS)=TABELLE$(VON+140,BIS+140)
25310 NEXT INDEX
25320 NEXT SATZ
25330 ZEIGER=Z-1
25340 GOTO 1000
```

Zum Löschen einer Adresse reicht es nicht, den entsprechenden Tabellenplatz zu löschen. Würden wir dies tun, so bleibt die Größe der Tabelle trotz Löschen einer Adresse erhalten. Dies ist nicht im Sinne des Erfinders. Alle Adressen, die sich hinter der zu löschenden Adresse befinden, müssen eine Stelle nach vorne rücken. Wenn z.B. die Adresse mit dem Index 5 gelöscht wird und 7 Adressen im Rechner gespeichert sind, so rückt die 6.

Adresse auf die 5., die somit gelöscht wird. Die letzte, also 7. Adresse rückt auf die 6. Wenn wir nun die Anzahl der Adressen auf 6, also um eins vermindern, so ist die Adresse mit dem Index 5 "echt" gelöscht worden.

Bei der Programmierung dieses Prinzips taucht zum ersten Mal eine verschachtelte Schleife auf. Innerhalb der Schleife mit der Variablen *'SATZ'* befindet sich die Schleife *'INDEX'*. Die Schleife *'SATZ'* durchläuft alle Datensätze ab dem zu löschenden Datensatz. Die Felder werden alle einzeln nachgerückt. Sind alle Felder aufgerückt, so kann der nächste Datensatz bearbeitet werden. Die äußere Schleife läuft nur bis *'Z-1'*, also bis zum letzten eingegebenen Datensatz, da der letzte Datensatz in Zeile 25300 mit *'VON+140,BIS+140'* angesprochen wird. Wenn Sie die entsprechenden Zeilen studieren, so erkennen Sie die Technik dieses "Aufrückens".

Nun haben wir auch diesen Teil unseres zur Zeit recht umfangreichen Programms fertiggestellt. Es folgt nun wie immer die komplette Auflistung dieses Abschnittes:

```
25000 REM ====================
25010 REM  ADRESSEN LOESCHEN
25020 REM ====================
25030 GOSUB 500
25035 Z=ZEIGER
25036 V$(1)=" ":V$(20)=" ":V$(2)=V$
25037 N$(1)=" ":N$(20)=" ":N$(2)=N$
25040 POSITION 1,10
25050 PRINT "VORNAME:     ";:INPUT V$
25060 PRINT "NAME:        ";:INPUT N$
25070 FOR ZEIGER=1 TO Z
25080 INDEX=2:GOSUB 600
25090 IF V$<>TABELLE$(VON,VON+LEN(V$)-1))THEN 25140
25100 INDEX=3:GOSUB 600
25110 IF N$<>TABELLE$(VON,VON+LEN(N$)-1))THEN 25140
25120 POP:GOTO 25150
25130 NEXT ZEIGER
25140 FEHLER$="ADRESSE NICHT VORHANDEN!":GOSUB 200:GOTO 1000
25150 GOSUB 300:POSITION 2,10
```

```
25160 FOR INDEX=1 TO 7
25170 GOSUB 600:PRINT FELD$(INDEX*10-9,INDEX*10);
25180 NEXT INDEX
25190 PRINT TABELLE$(VON,BIS)
25200 NEXT INDEX
25210 POSITION 1,18
25220 PRINT "ADRESSE LOESCHEN (J/N) ";
25230 INPUT ANTWORT$
25240 IF ANTWORT$="J" THEN 25270
25250 IF ANTWORT$="N" THEN ZEIGER=Z:GOTO 1000
25260 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 25210
25270 FOR SATZ=ZEIGER TO Z-1
25280 FOR INDEX=1 TO 7
25290 ZEIGER=SATZ: GOSUB 600
25300 TABELLE$(VON,BIS)=TABELLE$(VON+140,BIS+140)
25310 NEXT INDEX
25320 NEXT SATZ
25330 ZEIGER=Z-1
25340 GOTO 1000
```

ADRESSEN AUSGEBEN

Dieser Programmabschnitt gehört sicherlich nicht zu den leichtesten. Hier wollen wir zum ersten Mal auf dem Drucker ausgeben. Doch auch Bildschirmausgabe soll ermöglicht werden. Nach Auswahl, ob denn nun auf dem Bildschirm oder dem Drucker ausgegeben werden soll, geben Sie die Suchbegriffe ein. Doch schreiben wir zunächst die ersten Zeilen.

```
30000 REM ====================
30010 REM  ADRESSEN AUSGEBEN
30020 REM ====================
30030 GOSUB 500
30040 POSITION 1,10
30050 PRINT "DRUCKER ODER BILDSCHIRM (D/B) ";
30055 INPUT ANTWORT$
30060 IF ANTWORT$="D" THEN OPEN #1,8,0,"P:":GOTO 30090
30070 IF ANTWORT$="B" THEN OPEN #1,8,0,"E:":GOTO 30090
30080 FEHLER$="'D' ODER 'B' DRUECKEN!":GOSUB 200:GOTO 30040
```

Hier wird wieder geprüft, ob Daten im Rechner sind (Zeile 30030). Danach wird gefragt, ob auf Drucker oder Bildschirm ausgegeben werden soll. Die Antwort ('D' oder 'B') wird in der Stringvariablen '*ANTWORT$*' festgehalten. Falls der Drucker ausgewählt wurde, muß der "*Kanal*" zum Drucker geöffnet werden. Wenn der Bildschirm ausgewählt wurde, so wird auch hier ein "*Kanal*" geöffnet. Zwar kann die Ausgabe auf dem Bildschirm ohne einen '*OPEN*'-Befehl erfolgen, jedoch verwenden wir zur späteren Ausgabe eine andere Form des PRINT-Befehls:

```
Problem:    Ausgabe auf verschiedene Geräte

Befehl:     PRINT #kan; ......

Parameter:  kan - Nummer des Gerätekanals (1-15)

Beispiel:   PRINT #8;"ADRESSENLISTE"
            Gibt den Text "ADRESSENLISTE" auf das
            Gerät aus, das mit der Kanalnummer 8
            geöffnet wurde
```

Wir können also mit *PRINT* auch auf dem Drucker oder auf das Kassettenlaufwerk Daten ausgeben. Wir haben, abhängig von dem ausgewählten Gerät, den Bildschirm oder den Drucker geöffnet. Wird nun zur Ausgabe immer der Befehl *'PRINT #kan; ......'* angegeben, so wird immer auf das entsprechende Gerät ausgegeben.

Sicher möchten Sie genauer wissen, welche Einsatzmöglichkeiten der *OPEN*-Befehl hat. Die folgende Übersicht soll Sie aufklären:

```
Problem:    Vorbereitung der Ein/Ausgabe

Befehl:     OPEN #kan,betr,zus,ger

Parameter:  kan  - Nummer des Kanals (1-15)
            betr - Betriebsart
            zus  - Zusatzparameter (wird kaum genutzt)
            ger  - Gerätekurzzeichen

Beispiel:   OPEN 1,8,0,"P:"
            öffnet den Drucker (P:) zum Schreiben (8)
            über Kanal 1
```

Dies sagt noch nicht viel aus. Sie müssen die einzelnen Parameter einzusetzen wissen. Es folgt zunächst die Tabelle der wichtigsten *OPEN*-Parameter:

Gerät	Betriebsart	'betr'	'zus'	'ger'
Programm-recorder	Lesen	4	0	C:
	Schreiben	8	0	C:
Disketten-station	Lesen	4	0	D:
	Lesen des Directory	6	0	D:
	Schreiben neues File	8	0	D:
	Schreiben, erweitern	9	0	D:
	Lesen und Schreiben	12	0	D:
Bildschirm-editor	Schreiben	8	0	E:
Tastatur	Lesen	4	0	K:
Drucker	Schreiben	8	0	P:
Bildschirm	Schreiben	8	0	S:

Doch zurück zu unserem Programm. Da wir nicht immer grundsätzlich alle Adressen ausgeben möchten, wollen wir für jedes Feld ein Suchkriterium bestimmen. Schauen Sie sich zunächst die folgenden Zeilen an:

```
30090 GOSUB 300:POSITION 1,10
30100 PRINT "Suchbegriffe:"
30110 PRINT "-------------"
30120 PRINT
30125 Z=ZEIGER
```

Hier wird die Eingabe der Suchbegriffe eingeleitet und der Zeiger auf den letzten Datensatz gesichert. Es sollen nun bis zu sieben Suchkriterien eingegeben werden, nach denen dann im Adressenbestand gesucht werden soll. Die Suchbegriffe sollen in einer Tabelle (*SUCH$*) gespeichert werden. Diese Tabelle umfaßt 7 Einträge mit je 20 Zeichen, also 140 Zeichen. Richten wir diese Tabelle nun am Programmanfang ein:

```
27 DIM SUCH$(140)
```

Es geht weiter im Programm mit der Schleife, die die sieben Suchkriterien einlesen soll. Doch vorher wollen wir in diesem Programmteil die Tabelle *SUCH$* mit Leerzeichen auffüllen, da sie noch Zeichen aus vorangegangenen Suchen beinhalten könnte:

```
30130 SUCH$(1)=" ":SUCH$(140)=" ":SUCH$(2)=SUCH$
```

Die nächsten Programmzeilen stellen eine Schleife dar, die neben den Feldbezeichnungen die Suchkriterien einliest:

```
30135 FOR INDEX=1 TO 7
30140 PRINT FELD$(INDEX*10-9,INDEX*10);
30150 INPUT ZW$
30160 SUCH$(INDEX*20-19,INDEX*20)=ZW$
30170 NEXT INDEX
```

In der Schleife werden zuerst die Feldbezeichnungen ausgegeben. Die Begiffe werden dann in der Zeile 30150 eingelesen. Wenn z.B. alle Adressen aus Düsseldorf gesucht werden sollen, so geben Sie in den ersten 4 Feldern nur *RETURN* ein. Das 5. Feld, das die Postleitzahl und den Ort enthält, wird mit "*4000 DUESSELDORF*" gefüllt und anschließend mit RETURN abgeschlossen. Die restlichen 2 Felder werden wiederum mit *RETURN* ignoriert. Wollen Sie aber alle Herren aus Düsseldorf suchen, so geben Sie zusätzlich bei der Anrede "*HERR*" ein. Sie können also nach mehreren Begriffen gleichzeitig suchen.

Nach der Eingabe der Suchbegriffe soll die Suche beginnen.

```
30180 FOR ZEIGER=1 TO Z
30190 GEFUNDEN=0
30200 FOR INDEX=1 TO 7
30210 ZW$=SUCH$(INDEX*20-19,INDEX*20)
30215 IF ZW$(1,5)="     " THEN GEFUNDEN=GEFUNDEN+1:GOTO 30230
30216 GOSUB 600
30220 IF ZW$=TABELLE$(VON,VON+LEN(ZW$)-1) THEN GEFUNDEN=GEFUNDEN+1
30230 NEXT INDEX
```

Wir benötigen wieder eine zweifach verschachtelte Schleife, die innerhalb der Datensätze alle Felder durchsucht. Nach dem Anfang der äußeren Schleife (Zeile 30180) wird in Zeile 30190 eine Variable '*GEFUNDEN*' auf 0 gesetzt. In dieser Variablen sollen alle Felder gezählt werden, für die entweder kein Suchbegriff eingegeben wurde oder für die ein Suchbegriff eingegeben wurde, der mit dem Inhalt des Feldes übereinstimmt. Wenn für ein Feld kein Suchbegriff eingegeben, also nur *RETURN* gedrückt wurde, so enthält der Suchbegriff 20 Leerzeichen. Es werden dann jedoch nur die ersten fünf Stellen auf Leerzeichen abgefragt. Dieser Vergleich und das Erhöhen der Variablen '*GEFUNDEN*' geschieht in den Zeilen 30215 und 30220. Wenn nach Ablauf der inneren Schleife (*INDEX*) der Erfolgszähler '*GEFUNDEN*' 7 beträgt, entspricht die Adresse den Suchkriterien und kann ausgegeben werden. Die folgenden Zeilen leiten diese Ausgabe ein:

```
30240 IF GEFUNDEN<>7 THEN 30350
30250 IF ANTWORT$="B"THEN GOSUB 300
30260 PRINT #1
```

Hier wird die Ausgabe des gefundenen Datensatzes vorbereitet. Nur wenn der Datensatz nicht den Suchbegriffen entspricht ('*GEFUNDEN*' ungleich 7) wird die gesamte Ausgabe übersprungen. Wenn der Bildschirm ausgewählt wurde, wird ein neuer Kopf erzeugt. Anschließend wird auf das ausgewählte Gerät eine Leerzeile ausgegeben. Es folgen nun die restlichen Zeilen dieses Programmabschnittes, die anschließend wieder dokumentiert werden.

```
30270 Z1=INDEX:PRINT
30275 FOR I=1 TO 7
30280 PRINT #1;FELD$(I*10-9,I*10);
30290 INDEX=I:GOSUB 600
30300 PRINT #1;TABELLE$(VON,BIS)
30310 NEXT I:PRINT
30320 INDEX=Z1
30330 IF ANTWORT$="D" THEN 30350
30340 PRINT "DRUECKEN SIE RETURN!";:INPUT V$
30350 NEXT ZEIGER
30360 GOSUB 300:POSITION 18,1
30370 PRINT "** DATEIENDE **"
30380 PRINT "DRUECKEN SIE RETURN";:INPUT V$
30390 ZEIGER=Z:CLOSE #1: GOTO 1000
```

In diesen Zeilen dreht sich fast alles um die Ausgabe des gefundenen Datensatzes. Die Zeilen 30275 bis 30310 bilden eine Schleife, in der die 7 Felder der Adresse mit ihrer Bezeichnung auf das ausgewählte Gerät ausgegeben werden.

Wenn Bildschirmausgabe ausgewählt wurde, wird erst nach Drücken von *RETURN* mit der Suche fortgefahren (Zeile 30340). Weitersuchen heißt hier, die Schleife '*ZEIGER* mit dem Befehl '*NEXT ZEIGER*' um eins zu erhöhen. Ist die Druckerausgabe aktuell, so wird die Zeile zum Betätigen von *RETURN* übersprungen. Es werden also alle Adressen hintereinander ausgedruckt.

Hinter dem Ende der Schleife '*ZEIGER*' ist die Suche beendet. Es wird durch die Meldung "*** DATEIENDE ***" signalisiert. Wenn der Anwender nun *RETURN* drückt, wird der Programmabschnitt beendet, also zurück zum Menü verzweigt. Vorher wird der Kanal 1 noch geschlossen. Der Befehl '*CLOSE #1*' ist dafür zuständig.

Dies war ein weiterer Meilenstein zu unserer Adressenverwaltung. Nun müssen nur noch die Teile zum Laden und Speichern der Datei auf Kassette geschrieben werden.

Mit dem folgenden Listing können Sie nochmals Ihren Programmteil "ADRESSEN AUSGEBEN" kontrollieren:

```
30000 REM ====================
30010 REM  ADRESSEN AUSGEBEN
30020 REM ====================
30030 GOSUB 500
30040 POSITION 1,10
30050 PRINT "DRUCKER ODER BILDSCHIRM (D/B) ";
30055 INPUT ANTWORT$
30060 IF ANTWORT$="D" THEN OPEN #1,8,0,"P:":GOTO 30090
30070 IF ANTWORT$="B" THEN OPEN #1,8,0,"E:":GOTO 30090
30080 FEHLER$="'D' ODER 'B' DRUECKEN!":GOSUB 200:GOTO 30040
30090 GOSUB 300:POSITION 1,10
30100 PRINT "Suchbegriffe:"
30110 PRINT "-------------"
30120 PRINT
30125 Z=ZEIGER
30130 SUCH$(1)=" ":SUCH$(140)=" ":SUCH$(2)=SUCH$
30135 FOR INDEX=1 TO 7
30140 PRINT FELD$(INDEX*10-9,INDEX*10);
30150 INPUT ZW$
30160 SUCH$(INDEX*20-19,INDEX*20)=ZW$
30170 NEXT INDEX
30180 FOR ZEIGER=1 TO Z
30190 GEFUNDEN=0
30200 FOR INDEX=1 TO 7
30210 ZW$=SUCH$(INDEX*20-19,INDEX*20)
30215 IF ZW$(1,5)="     " THEN GEFUNDEN=GEFUNDEN+1:GOTO 30230
30216 GOSUB 600
30220 IF ZW$=TABELLE$(VON,BIS) THENGEFUNDEN=GEFUNDEN+1
30230 NEXT INDEX
30240 IF GEFUNDEN<>7 THEN 30350
30250 IF ANTWORT$="B"THEN GOSUB 300
30260 PRINT #1
30270 Z1=INDEX
30275 FOR I=1 TO 7
30280 PRINT #1,FELD$(I*10-9,I*10);
30290 INDEX=I:GOSUB 600
30300 PRINT #1,TABELLE$(VON,BIS)
```

```
30310 NEXT I
30320 INDEX=Z1
30330 IF ANTWORT$="D" THEN 30350
30340 PRINT "DRUECKEN SIE RETURN!";:INPUT V$
30350 NEXT ZEIGER
30360 GOSUB 300:POSITION 18,1
30370 PRINT "** DATEIENDE **"
30380 PRINT "DRUECKEN SIE RETURN";:INPUT V$
30390 ZEIGER=Z:CLOSE #1:GOTO 1000
```

DATEI SICHERN

Wir nähern uns dem Abschluß der Adressenverwaltung. Die eingegebenen Daten gehen mit dem Abschalten des Rechners natürlich verloren. Wir wollen unsere Adressen auf dem Kassettenlaufwerk speichern. Die Unterprogramme zum Laden und zum Speichern der Daten auf dem Diskettenlaufwerk finden Sie im Anhang dieses Buches. Was muß nun zuerst beachtet werden? Beginnen wir diesen Teil wie üblich mit dem Bildschirmkopf und der Abfrage, ob Daten im Rechner gespeichert sind.

```
10000 REM ====================
10010 REM   DATEI SPEICHERN
10020 REM ====================
10030 GOSUB 500
```

Diese Zeilen sind uns nicht mehr unbekannt. Bevor wir die Daten auf der Kassette speichern können, muß diese zur Speicherung vorbereitet werden. Die folgenden *PRINT*-Befehle weisen den Benutzer des Programms darauf hin:

```
10040 POSITION 2,10
10050 PRINT "FOLGENDE VORBEREITUNGEN DURCHFUEHREN:"
10060 PRINT "-------------------------------------"
10070 PRINT
10080 PRINT "1. KASSETTE ZUM SPEICHERN EINLEGEN"
10090 PRINT "2. KASSETTE ZURUECKSPULEN"
10100 PRINT "3. NACH SIGNAL AUFNAHMETASTEN DRUECKEN"
10110 PRINT "4. TASTE RETURN BETAETIGEN"
10120 PRINT
10130 POSITION 5,18
10140 PRINT "FERTIG (J) ";
10150 INPUT ANTWORT$
10160 IF ANTWORT$="J" THEN 101803
10170 FEHLER$="TASTE (J) DRUECKEN!":GOSUB 200:GOTO 101303
```

Auch diese Zeilen dürften Ihnen keine Verständnisschwierigkeiten bereiten. Doch nun geht es los. Um die Daten mit dem Befehl '*PRINT #1;...*' auf Kassette speichern zu können, müssen wir das Kassettenlaufwerk vorbereiten. Man sagt dazu, "die Datei muß geöffnet werden". Dazu setzen wir den bereits bekannten Befehl '*OPEN*' ein:

```
10180 OPEN #1,8,0,"C:ADRESSEN"
```

Mit diesem Befehl öffnen wir also unsere Datei, bevor die Daten übermittelt werden. Sie sehen hier, daß neben der Gerätebezeichnung noch ein Dateiname angegeben werden kann. Wird hier ein Name angegeben, so lädt die LADE-Routine nicht die ersten Datei, die auf der Kassette enthalten ist, sondern sucht nach der Datei mit dem Namen "*ADRESSEN*".

Die Daten können jetzt gespeichert werden. Das erste, was wir auf die Kassette schreiben, ist die Anzahl der zu übermittelnden Datensätze, die bekanntlich in '*ZEIGER*' gespeichert ist. Danach senden wir die einzelnen Felder der Adressentabelle mit Hilfe einer verschachtelten Schleife, die Ihnen nicht mehr unbekannt ist. Zuerst Feld 1 bis 7 des ersten Datensatzes, dann Feld 1 bis 7 des zweiten, usw.:

```
10190 PRINT #1;ZEIGER
10195 Z=ZEIGER
10200 FOR ZEIGER=1 TO Z
10210 FOR INDEX=1 TO 7
10220 GOSUB 600
10230 PRINT #1;TABELLE$(VON,BIS)
10240 NEXT INDEX
10250 NEXT ZEIGER
```

Da die Routine ab Zeile 600, die bekanntlich Anfangs- und Endposition der Adressenfelder ermittelt, die Variablen *'ZEIGER'* und *'INDEX'* benötigt, müssen wir diese Variablen hier auch als Schleifenvariablen einsetzen. Innerhalb der zweifach verschachtelten Schleife wird dann zunächst die Feldposition (Zeile 10220) ermittelt und anschließend das Feld übertragen.

Nach der Speicherung der Daten muß die Datei wieder geschlossen werden, bevor wieder zum Menü verzweigt werden kann:

```
10260 CLOSE #1
```

Danach soll noch eine Meldung ausgegeben werden:

```
10270 GOSUB 300
10280 PRINT "DIE ADRESSEN SIND GESPEICHERT!"
10290 FOR I=1 TO 1000:NEXT I
10300 GOTO 1000
```

Zunächst wird ein neuer Bildschirm erzeugt. Nach dem Hinweis folgt eine Warteschleife von ca. 2 Sekunden. Danach wird ins Menü verzweigt.

So einfach ist das Speichern von Daten auf dem Kassettenlaufwerk. Hier noch einmal das gesamte Teilprogramm:

```
10000 REM ====================
10010 REM   DATEI SPEICHERN
10020 REM ====================
10030 GOSUB 500
10040 POSITION 2,10
10050 PRINT "FOLGENDE VORBEREITUNGEN DURCHFUEHREN:"
10060 PRINT "-------------------------------------"
10070 PRINT
10080 PRINT "1. KASSETTE ZUM SPEICHERN EINLEGEN"
10090 PRINT "2. KASSETTE ZURUECKSPULEN"
10100 PRINT "3. NACH SIGNAL AUFNAHMETASTEN DRUECKEN"
10110 PRINT "4. TASTE RETURN BETAETIGEN"
10120 PRINT
10130 POSITION 5,18
10140 PRINT "FERTIG (J) ";
10150 INPUT ANTWORT$
10160 IF ANTWORT$="J" THEN 10180
10170 FEHLER$="TASTE (J) DRUECKEN!":GOSUB 200:GOTO 10130
10180 OPEN #1,8,0,"C:ADRESSEN"
10190 PRINT #1;ZEIGER
10195 Z=ZEIGER
10200 FOR ZEIGER=1 TO Z
10210 FOR INDEX=1 TO 7
10220 GOSUB 600
10230 PRINT #1;TABELLE$(VON,BIS)
10240 NEXT INDEX
10250 NEXT ZEIGER
10260 CLOSE #1
10270 GOSUB 300
10280 PRINT "DIE ADRESSEN SIND GESPEICHERT!"
10290 FOR I=1 TO 1000:NEXT I
10300 GOTO 1000
```

DATEI LADEN

Das Laden der Adressen verläuft fast auf die gleiche Weise wie das Speichern. Hier wird nicht geprüft, ob Daten im Rechner vorhanden sind. Der Anwender sollte also neu eingegebenen Adressen erst abspeichern, bevor er eine Adressendatei lädt.

```
5000 REM ====================
5010 REM   DATEI LADEN
5020 REM ====================
5040 POSITION 2,10
5050 PRINT "FOLGENDE VORBEREITUNGEN DURCHFUEHREN:"
5060 PRINT "-------------------------------------"
5070 PRINT
5080 PRINT "1. KASSETTE ZUM LADEN EINLEGEN"
5090 PRINT "2. KASSETTE ZURUECKSPULEN"
5100 PRINT "3. NACH SIGNAL PLAY-TASTE DRUECKEN"
5110 PRINT "4. TASTE RETURN BETAETIGEN"
5120 PRINT
5130 POSITION 5,18
5140 PRINT "FERTIG (J) ";
5150 INPUT ANTWORT$
5160 IF ANTWORT$="J" THEN 5180
5170 FEHLER$="TASTE (J) DRUECKEN!":GOSUB 200:GOTO 5130
```

Nachdem alle Vorbereitungen getroffen sind, kann die Datei wieder geöffnet werden. Diesmal aber zum Lesen. Dazu wird die Betriebsart des *OPEN*-Befehl auf 4 gesetzt:

```
5180 OPEN #1,4,0,"C:ADRESSEN"
```

Nur benötigen wir wieder eine zweifach verschachtelte Schleife zum Einlesen der Adressen:

```
5190 INPUT #1;Z
5200 FOR ZEIGER=1 TO Z
5210 FOR INDEX=1 TO 7
5220 GOSUB 600
5225 INPUT #1;ZW$
5230 TABELLE$(VON,BIS)=ZW$
5240 NEXT INDEX
5250 NEXT ZEIGER
5260 CLOSE #1
5270 GOSUB 300
5280 PRINT "DIE ADRESSEN SIND GELADEN!"
5290 FOR I=1 TO 1000:NEXT I
5300 GOTO 1000
```

Es wird zunächst die Anzahl der in dieser Datei gespeicherten Daten eingelesen. Diese Variable wird als Endwert für die Schleife '*ZEIGER*' benutzt. Es werden also soviel Datensätze eingelesen, wie zuvor gespeichert wurden. Der Kanal wird in Zeile 5260 mit dem Befehl '*CLOSE #1*' wieder geschlossen.

Die letzten Zeilen dieses Abschnitts geben wieder eine Meldung aus und verzweigen nach der Warteschleife zurück ins Menü.

PROGRAMM BEENDEN

Wie zu Anfang des Kapitels bereits erwähnt, sollte ein Programm nicht mit Hilfe des Netzschalters oder mit der *RESET-TASTE* ausgeschaltet werden. Wie ein Programm beendet werden sollte, wird nun demonstriert. Blicken wir zunächst wieder auf die ersten Zeilen.

```
31000 REM ====================
31010 REM   PROGRAMM BEENDEN
31020 REM ====================
31030 IF ZEIGER=0 THEN 31150
```

Die ersten Zeilen werden Ihnen sicher schon langweilig. Doch die Zeile 31030 sieht ungewohnt aus. Wenn *'ZEIGER'* 0 ist, also wenn keine Daten im Rechner gespeichert sind, wird das Programm ohne weiteres sofort beendet. Dies geschieht in Zeile 35150.

```
31040 POSITION 2,12
31050 PRINT"SIND ALLE DATEN GESICHERT (J/N) ";
31060 INPUT ANTWORT$
31070 IF ANTWORT$="N" THEN 1000
31080 IF ANTWORT$="J" THEN 31100
31090 FEHLER$="J ODER N DRUECKEN!":GOSUB 200:GOTO 31040
```

Langsam erkennt man den Sinn eines Programmteils "*PROGRAMM BEENDEN*". Es wird also festgestellt, ob alle Daten gesichert wurden. Immerhin kann man diesen Programmabschnitt versehentlich aufgerufen haben. Wenn Sie nun mit nein antworten, kommen Sie wieder in das Menü zurück. Wenn Sie die Adressen jedoch abgespeichert haben, kann Sie nichts daran hindern, das Programm zu beenden.

```
31100 GOSUB 300:POSITION 12,1
31110 PRINT"DAS PROGRAMM KANN MIT 'GOTO 1000' WIEDER"
31120 PRINT"GESTARTET WERDEN, OHNE DASS DATEN VER-"
31130 PRINT"LOREN GEHEN!!"
31140 PRINT
31150 END
```

Nun folgt vor Beendigung des Programms ein wichtiger Hinweis: Nach dem Verlassen des Programms kann man es mit dem Befehl 'GOTO 1000' wieder ohne Datenverlust starten. Der Befehl RUN löscht vor dem Start sämtliche Variablen, ist also nicht zu empfehlen.

Jetzt haben wir es geschafft. Sie besitzen nun eine bis aufs letzte dokumentierte Adressenverwaltung, bei deren Erstellung Sie sicher viel gelernt haben. Das, was Sie nun gelernt haben, können Sie entweder zur Verwirklichung eigener Ideen oder zur Durchführung individueller Programmänderungen, speziell bei dieser Adressenverwaltung, verwenden. Falls Sie einige Abschnitte nicht sofort verstanden haben, arbeiten Sie diese ruhig nochmals durch.

ADRESSENVERWALTUNG KOMPLETT

```
1 REM --------------------
2 REM   STRINGVARIABLEN
3 REM --------------------
4 DIM FEHLER$(38)
5 DIM TABELLE$(14000)
6 TABELLE$(1)=" ":TABELLE$(14000)=" "
7 TABELLE$(2)=TABELLE$
8 DIM FUNKTION$(119)
9 FUNKTION$(1,17)    ="   DATEI LADEN   "
10 FUNKTION$(18,34)  =" DATEI SPEICHERN "
11 FUNKTION$(35,51)  ="ADRESSEN EINGEBEN"
12 FUNKTION$(52,68)  ="ADRESSEN AENDERN "
13 FUNKTION$(69,85)  ="ADRESSEN LOESCHEN"
14 FUNKTION$(86,102) ="ADRESSEN AUSGEBEN"
15 FUNKTION$(103,119)="PROGRAMM BEENDEN "
16 DIM FELD$(70)
17 FELD$(1,10)  ="ANREDE    "
18 FELD$(11,20) ="VORNAME   "
19 FELD$(21,30) ="NAME      "
20 FELD$(31,40) ="STRASSE   "
21 FELD$(41,50) ="PLZ/ORT   "
22 FELD$(51,60) ="TELEFON   "
23 FELD$(61,70) ="BEMERKUNG "
24 DIM ZW$(20)
25 DIM ANTWORT$(1)
26 DIM V$(20),N$(20)
27 DIM SUCH$(140)
99 GOTO 1000

100 REM ====================
110 REM    Programmkopf
120 REM ====================
130 PRINT CHR$(125)
140 FOR I=1 TO 38:PRINT "=";:NEXT I
150 POSITION 12,2
160 PRINT "ADRESSENVERWALTUNG"
170 FOR I=1 TO 38:PRINT "=";:NEXT I
180 RETURN
```

```
200 REM ===================
210 REM    FEHLERMELDUNG
220 REM ===================
230 POSITION 1,23
240 PRINT FEHLER$;
250 PRINT CHR$(253);CHR$(253);:REM 2 MAL PIEP
260 POSITION 1,23
270 PRINT CHR$(156);:REM ZEILE LOESCHEN
280 RETURN

300 REM ====================
310 REM KOEPFE PROGRAMMTEILE
320 REM ====================
330 GOSUB 100
340 POSITION 10,5
350 FOR I=1 TO 19:PRINT"*";:NEXT I
360 POSITION 10,6
370 PRINT "*";FUNKTION$(FUNKTION*17-16,FUNKTION*17);"*"
380 POSITION 10,7
390 FOR I=1 TO 19:PRINT"*";:NEXT I
395 RETURN

500 REM ====================
510 REM     KEINE DATEN!
520 REM ====================
530 IF ZEIGER>0 THEN RETURN
540 FEHLER$="KEINE DATEN IM RECHNER!"
550 GOSUB 200
560 POP:GOTO1000

600 REM ====================
610 REM  ADRESSEN-POSITION
620 REM ====================
630 VON=(ZEIGER*140-139)+(INDEX*20-20)
640 BIS=VON+19
650 RETURN
```

```
1000 REM ====================
1010 REM    HAUPTPROGRAMM
1020 REM ====================
1030 GOSUB 100
1040 POSITION 1,7
1050 PRINT"     PROGRAMMFUNKTIONEN:"
1060 PRINT"     ------------------"
1070 PRINT
1080 PRINT"     -1- DATEI LADEN"
1090 PRINT"     -2- DATEI SPEICHERN"
1100 PRINT"     -3- ADRESSEN EINGEBEN"
1110 PRINT"     -4- ADRESSEN AENDERN"
1120 PRINT"     -5- ADRESSEN LOESCHEN"
1130 PRINT"     -6- ADRESSEN AUSGEBEN"
1140 PRINT"     -7- PROGRAMM BEENDEN"
1150 POSITION 10,18:PRINT"AUSWAHL: 0 ";
1160 POSITION 18,18:INPUT FUNKTION
1170 IF FUNKTION=0 OR FUNKTION>7 THEN FEHLER$="UNGUELTIGER WERT"
  :GOSUB 200:GOTO 1160
1175 GOSUB 300
1180 ON FUNKTION GOTO 5000,10000,15000,20000,25000,30000,31000

5000 REM ====================
5010 REM    DATEI LADEN
5020 REM ====================
5040 POSITION 2,10
5050 PRINT "FOLGENDE VORBEREITUNGEN DURCHFUEHREN:"
5060 PRINT "-------------------------------------"
5070 PRINT
5080 PRINT "1. KASSETTE ZUM LADEN EINLEGEN"
5090 PRINT "2. KASSETTE ZURUECKSPULEN"
5100 PRINT "3. NACH SIGNAL PLAY-TASTE DRUECKEN"
5110 PRINT "4. TASTE RETURN BETAETIGEN"
5120 PRINT
5130 POSITION 5,18
5140 PRINT "FERTIG (J) ";
5150 INPUT ANTWORT$
5160 IF ANTWORT$="J" THEN 5180
5170 FEHLER$="TASTE (J) DRUECKEN!":GOSUB 200:GOTO 5130
5180 OPEN #1,4,0,"C:ADRESSEN"
```

```
5190 INPUT #1;Z
5200 FOR ZEIGER=1 TO Z
5210 FOR INDEX=1 TO 7
5220 GOSUB 600
5225 INPUT #1;ZW$
5230 TABELLE$(VON,BIS)=ZW$
5240 NEXT INDEX
5250 NEXT ZEIGER
5260 CLOSE #1
5270 GOSUB 300
5280 PRINT "DIE ADRESSEN SIND GELADEN!"
5290 FOR I=1 TO 1000:NEXT I
5300 GOTO 1000

10000 REM ====================
10010 REM    DATEI SPEICHERN
10020 REM ====================
10030 GOSUB 500
10040 POSITION 2,10
10050 PRINT "FOLGENDE VORBEREITUNGENDURCHFUEHREN:"
10060 PRINT "------------------------------------"
10070 PRINT
10080 PRINT "1. KASSETTE ZUM SPEICHERN EINLEGEN"
10090 PRINT "2. KASSETTE ZURUECKSPULEN"
10100 PRINT "3. NACH SIGNAL AUFNAHMETASTEN DRUECKEN"
10110 PRINT "4. TASTE RETURN BETAETIGEN"
10120 PRINT
10130 POSITION 5,18
10140 PRINT "FERTIG (J) ";
10150 INPUT ANTWORT$
10160 IF ANTWORT$="J" THEN 10180
10170 FEHLER$="TASTE (J) DRUECKEN!":GOSUB 200:GOTO 10130
10180 OPEN #1,8,0,"C:ADRESSEN"
10190 PRINT #1;ZEIGER
10195 Z=ZEIGER
10200 FOR ZEIGER=1 TO Z
10210 FOR INDEX=1 TO 7
10220 GOSUB 600
10230 PRINT #1;TABELLE$(VON,BIS)
10240 NEXT INDEX
```

```
10250 NEXT ZEIGER
10260 CLOSE #1
10270 GOSUB 300
10280 PRINT "DIE ADRESSEN SIND GESPEICHERT!"
10290 FOR I=1 TO 1000:NEXT I
10300 GOTO 1000

15000 REM ====================
15010 REM  ADRESSEN EINGEBEN
15020 REM ====================
15030 ZEIGER=ZEIGER+1
15040 PRINT
15050 FOR INDEX=1 TO 7
15055 GOSUB 600
15060 PRINT FELD$(INDEX*10-9,INDEX*10)
15065 INPUT ZW$
15070 TABELLE$(VON,BIS)=ZW$
15080 NEXT INDEX
15090 POSITION 2,22
15100 PRINT "DATEN RICHTIG EINGEGEBEN (J/N)";
15110 INPUT ANTWORT$
15120 IF ANTWORT$="J"THEN 15150
15130 IF ANTWORT$="N" THEN ZEIGER=ZEIGER-1:GOSUB 300:GOTO 15000
15140 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 15090
15150 POSITION 2,23
15160 PRINT "WEITERE EINGABEN (J/N)";
15170 INPUT ANTWORT$
15180 IF ANTORT$="J" THEN GOSUB 300:GOTO 15000
15190 IF ANTWORT$="N" THEN 1000
15200 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 15150

20000 REM ====================
20010 REM   ADRESSEN AENDERN
20020 REM ====================
20025 GOSUB 500
20030 ZAEHLER=1:Z=ZEIGER
20040 POSITION 2,10
20050 FOR INDEX=1 TO 7
20055 ZEIGER=ZAEHLER:GOSUB 600
20060 PRINT INDEX;"-";FELD$(INDEX*10-9,INDEX*10);
```

```
20065 PRINT TABELLE$(VON,BIS)
20070 NEXT INDEX
20075 POSITION 5,18
20080 PRINT "KOMMANDO (+ - * M)";
20085 INPUT ANTWORT$
20090 IF ANTWORT$="M"THEN ZEIGER=Z:GOTO 1000
20100 IF ANTWORT$="+" AND ZAEHLER<Z THEN ZAEHLER=ZAEHLER+1:GOTO 20040
20110 IF ANTWORT$="-" AND ZAEHLER>1 THEN ZAEHLER=ZAEHLER-1:GOTO 20040
20120 IF ANTWORT$="*" THEN 20140
20130 FEHLER$="EINGABEFEHLER!":GOSUB 200:GOTO 20075
20140 POSITION 5,19
20145 PRINT "FELDNUMMER (1-7): 0 ";
20150 POSITION 22,19:INPUT INDEX
20155 IF INDEX>0 AND INDEX<8 THEN 20165
20160 FEHLER$="NUMMER 1-7 EINGEBEN!":GOSUB 200:GOTO 20150
20165 ZEIGER=ZAEHLER:GOSUB 600
20170 PRINT "NEUER INHALT: ";
20180 INPUT ZW$
20185 TABELLE$(VON,BIS)=ZW$
20190 GOSUB 300:GOTO 20040

25000 REM ====================
25010 REM  ADRESSEN LOESCHEN
25020 REM ====================
25030 GOSUB 500
25035 Z=ZEIGER
25036 V$(1)=" ":V$(20)=" ":V$(2)=V$
25037 N$(1)=" ":N$(20)=" ":N$(2)=N$
25040 POSITION 1,10
25050 PRINT "VORNAME:     ";:INPUT V$
25060 PRINT "NAME:        ";:INPUT N$
25070 FOR ZEIGER=1 TO Z
25080 INDEX=2:GOSUB 600
25090 IF V$<>TABELLE$(VON,VON+LEN(V$)-1) THEN 25140
25100 INDEX=3:GOSUB 600
25110 IF N$<>TABELLE$(VON,VON+LEN(N$)-1) THEN 25140
25120 POP:GOTO 25150
25130 NEXT ZEIGER
25140 FEHLER$="ADRESSE NICHT VORHANDEN!":GOSUB 200:GOTO 1000
25150 GOSUB 300:POSITION 2,10
```

```
25160 FOR INDEX=1 TO 7
25170 GOSUB 600:PRINT FELD$(INDEX*10-9,INDEX*10);
25180 NEXT INDEX
25190 PRINT TABELLE$(VON,BIS)
25200 NEXT INDEX
25210 POSITION 1,18
25220 PRINT "ADRESSE LOESCHEN (J/N) ";
25230 INPUT ANTWORT$
25240 IF ANTWORT$="J" THEN 25270
25250 IF ANTWORT$="N" THEN ZEIGER=Z:GOTO 1000
25260 FEHLER$="'J' ODER 'N' DRUECKEN!":GOSUB 200:GOTO 25210
25270 FOR SATZ=ZEIGER TO Z-1
25280 FOR INDEX=1 TO 7
25290 ZEIGER=SATZ: GOSUB 600
25300 TABELLE$(VON,BIS)=TABELLE$(VON+140,BIS+140)
25310 NEXT INDEX
25320 NEXT SATZ
25330 ZEIGER=Z-1
25340 GOTO 1000

30000 REM ====================
30010 REM  ADRESSEN AUSGEBEN
30020 REM ====================
30030 GOSUB 500
30040 POSITION 1,10
30050 PRINT "DRUCKER ODER BILDSCHIRM (D/B) ";
30055 INPUT ANTWORT$
30060 IF ANTWORT$="D" THEN OPEN #1,8,0,"P:":GOTO 30090
30070 IF ANTWORT$="B" THEN OPEN #1,8,0,"E:":GOTO 30090
30080 FEHLER$="'D' ODER 'B' DRUECKEN!":GOSUB 200:GOTO 30040
30090 GOSUB 300:POSITION 1,10
30100 PRINT "Suchbegriffe:"
30110 PRINT "-------------"
30120 PRINT
30125 Z=ZEIGER
30130 SUCH$(1)=" ":SUCH$(140)=" ":SUCH$(2)=SUCH$
30135 FOR INDEX=1 TO 7
30140 PRINT FELD$(INDEX*10-9,INDEX*10);
30150 INPUT ZW$
30160 SUCH$(INDEX*20-19,INDEX*20)=ZW$
```

```
30170 NEXT INDEX
30180 FOR ZEIGER=1 TO Z
30190 GEFUNDEN=0
30200 FOR INDEX=1 TO 7
30210 ZW$=SUCH$(INDEX*20-19,INDEX*20)
30215 IF ZW$(1,5)="     " THEN GEFUNDEN=GEFUNDEN+1:GOTO 30230
30216 GOSUB 600
30220 IF ZW$=TABELLE$(VON,BIS) THENGEFUNDEN=GEFUNDEN+1
30230 NEXT INDEX
30240 IF GEFUNDEN<>7 THEN 30350
30250 IF ANTWORT$="B"THEN GOSUB 300
30260 PRINT #1
30270 Z1=INDEX
30275 FOR I=1 TO 7
30280 PRINT #1,FELD$(I*10-9,I*10);
30290 INDEX=I:GOSUB 600
30300 PRINT #1,TABELLE$(VON,BIS)
30310 NEXT I
30320 INDEX=Z1
30330 IF ANTWORT$="D" THEN 30350
30340 PRINT "DRUECKEN SIE RETURN!";:INPUT V$
30350 NEXT ZEIGER
30360 GOSUB 300:POSITION 18,1
30370 PRINT "** DATEIENDE **"
30380 PRINT "DRUECKEN SIE RETURN";:INPUT V$
30390 ZEIGER=Z:CLOSE #1:GOTO 1000
```

```
31000 REM ====================
31010 REM   PROGRAMM BEENDEN
31020 REM ====================
31030 IF ZEIGER=0 THEN 31150
31040 POSITION 2,12
31050 PRINT"SIND ALLE DATEN GESICHERT (J/N) ";
31060 INPUT ANTWORT$
31070 IF ANTWORT$="N" THEN 1000
31080 IF ANTWORT$="J" THEN 31100
31090 FEHLER$="J ODER N DRUECKEN!":GOSUB 200:GOTO 31040
31100 GOSUB 300:POSITION 12,1
31110 PRINT"DAS PROGRAMM KANN MIT 'GOTO 1000' WIEDER"
31120 PRINT"GESTARTET WERDEN, OHNE DASS DATEN VER-"
31130 PRINT"LOREN GEHEN!!"
31140 PRINT
31150 END
```

KAPITEL 5: FARBE UND GRAFIK

Für viele Käufer der zahlreich angebotenen Homecomputer ist die Farbdarstellung und vor allem die Grafikfähigkeit des Rechners ein entscheidendes Kaufkriterium. Der ATARI bietet in seiner Preisklasse unvergleichbare Farbenpracht und hervorragende Grafikeigenschaften. So verfügt der Besitzer der ATARI über 16 Farben in jeweils 8 Hellkeitsstufen. Ihnen stehen also 128 Farbvariationen zur Verfügung. Die Auflösung der Grafik reicht bis zu 320 x 192 Bildpunkten, was für einen Rechner dieser Preisklasse als normal zu bezeichnen ist.

DIE GRAFIKBETRIEBSARTEN

Der ATARI arbeitet mit 16 verschiedenen Grafikbetriebsarten, die mit dem Befehl 'GRAPHICS' ausgewählt werden. Schauen Sie sich zunächst die Kurzbeschreibung dieses Befehls an:

```
Problem:    Auswahl der Grafikbetriebsarten

Befehl:     GRAPHICS mod+zus

Parameter:  mod - Betriebsart (0 bis 15)
            zus - Zusatzparameter

Beispiel:   GRAPHICS 8+24
            Schaltet um auf Grafikbetriebsart 8 (hoch-
            auflösende Grafik, ohne Textfenster (+24)
```

Was versteht man nun unter Grafikbetriebsarten? Diese Frage werden Sie nun bestimmt stellen. Das gesamte Gebiet der Grafik beim ATARI ist nicht einfach zu überschauen. Es ist notwendig, hier systematisch vorzugehen, ohne daß Sie den Überblick verlieren.

Beginnen wir mit den Grafikbetriebsarten, von denen Ihnen wie gesagt 16 verschiedene zur Verfügung stehen. Die folgende Tabelle soll Ihnen zunächst einen kleinen Überblick geben:

Grafikbe-triebsart	Typ	Spalten	Zeilen	Anzahl Farben
0	Text	40	24	1
1	Text	20	20/24	5
2	Text	20	10/12	5
3	Grafik	40	20/24	4
4	Grafik	80	40/48	2
5	Grafik	80	40/48	4
6	Grafik	160	80/96	2
7	Grafik	160	80/96	4
8	Grafik	320	160/192	1
9	Grafik	80	192	1
10	Grafik	80	192	9
11	Grafik	80	192	16
12	Grafik	40	20/24	5
13	Grafik	40	10/12	5
14	Grafik	160	160/192	2
15	Grafik	160	160/192	4

TEXTFENSTER

Sie haben sicher bemerkt, daß in der Spalte "Zeilen" zwei Werte angegeben sind. Die Begründung dafür ist folgende: Es besteht die Möglichkeit, die Anzahl der darstellbaren Zeilen zugunsten eines Textfensters einzuschränken. So kann man z.B. hochauflösende Grafikbefehle im Textfenster eingeben und das Resultat auf dem Grafikbildschirm betrachten. Arbeiten Sie ohne Textfenster, so müssen Sie immer zwischen Grafik- und Textbetriebsart hin- und herschalten.

Geben Sie den GRAPHICS-Befehl ohne Zusatz ein, so erscheint ein Textfenster. Überzeugen Sie sich davon, indem Sie den folgenden Befehl eingeben:

GRAPHICS 1

Sie erkennen nun deutlich, daß unten am Bildschirmrand vier Textzeilen angeordnet sind. Der Cursor läßt sich auch nicht aus diesen Bereich herausbewegen. Versuchen Sie es selbst. Wenn Sie ein Programm eingegeben haben und dieses in diesem Textfenster auflisten, so "rollt" das Programm über diese vier Zeilen. Das Editieren des Programms wird dadurch erschwert. Der Vorteil ist jedoch, daß Sie die Auswirkung eingegebener Grafikbefehle sofort erkennen.

Wenn dieses Textfenster nicht eingeblendet werden soll, so wird dem Befehl GRAPHICS ein Zusatzparameter angehängt.

ZU DER GRAFIKBETRIEBSART MUSS 16 ADDIERT WERDEN

Wenn Sie nun den Befehl

GRAPHICS 1+16

eingeben, werden Sie feststellen, daß nicht etwa das Fenster ausgeblendet, sondern zurück zur Betriebsart 0 geschaltet wurde. Der Grund dafür ist folgender:

DAS TEXTFENSTER KANN NICHT IM DIREKTMODUS AUSGESCHALTET WERDEN.

Dies ist auch verständlich, denn wie wollen Sie ohne Textfenster Befehle eingeben? Soll das Textfenster also ausgeblendet werden, so ist dies nur innerhalb eines Programms möglich. Das folgende Programm demonstriert dies:

```
10 GRAPHICS 1+16
20 FOR I=1 TO 1000:NEXT I
30 GRAPHICS 0
```

Geben Sie dieses Programm ein und beobachten Sie, was passiert. Das Programm schaltet für ca. 2 Sekunden die Grafikbetriebsart 1 ohne Textfenster ein und anschließend wieder in die Betriebsart 0 zurück.

Wie sieht es nun mit der Ausgabe des PRINT-Befehls aus?

PRINT GIBT STETS AUF DEM TEXTFENSTER AUS.

Es ist also nicht möglich, mit dem normalen PRINT-Befehl auf den Bildschirm auszugeben. Das folgende Programm demonstriert dies:

```
10 GRAPHICS 1+16
15 PRINT "ATARI"
20 FOR I=1 TO 1000:NEXT I
30 GRAPHICS 0
```

Wenn Sie dieses Programm starten, so "streikt" Ihr ATARI, indem er einfach zur Betriebsart 0 schaltet, da kein Textfenster vorhanden ist. Ändern wir das Programm nun wie folgt ab, so arbeitet es erwartungsgemäß:

```
10 GRAPHICS 1
15 PRINT "ATARI"
20 FOR I=1 TO 1000:NEXT I
30 GRAPHICS 0
```

Der Text "ATARI" wird ins Fenster geschrieben und das Programm kehrt anschließend wieder in die Betriebsart 0 zurück.

ZEICHEN DOPPELTER BREITE UND HÖHE

In den Grafikbetriebsarten 1 und 2 ist es möglich, vergrößerte Buchstaben auszugeben.

Betriebsart	Auswirkung
1	Buchstaben doppelter Breite (20 Zeichen auf 20 oder 24 Zeilen)
2	Buchstaben doppelter Breite und Höhe (20 Zeichen auf 10 oder 12 Zeilen)

Sicher fragen Sie sich nun, wie die Buchstaben denn auf dem Bildschirm dargestellt werden, da der PRINT-Befehl dazu ja nicht geeignet ist. "Kein Problem" würde ein ATARI-Freak nun sagen, dazu gibt es den Befehl 'PRINT #6'. Sie kennen bereits diese Art des PRINT-Befehls aus der Adressenverwaltung. Der Befehl wird wie folgt eingesetzt:

```
Problem:    Ausgabe vergrößerter Zeichen

Befehl:     PRINT #6;"....."

Beispiel:   GRAPHICS 1
            PRINT #6;"ATARI"
            Gibt den Text "ATARI" in Buchstaben
            doppelter Breite auf dem Bildschirm aus
```

Im folgenden Programm wird dieser Befehl eingesetzt:

```
5 DIM TEXT$(20)
10 GRAPHICS 1
20 PRINT "TEXT:";
30 INPUT TEXT$
40 PRINT #6;TEXT$
50 FOR I=1 TO 1000:NEXT I
60 GRAPHICS 0
```

Hier wird ein Text in das Textfenster eingelesen und anschließend auf dem Bildschirm ausgegeben. Dies ist auch in Betriebsart 2 mit Buchstaben in doppelter Breite und Höhe möglich. Ändern Sie dazu die Zeile 10:

```
10 GRAPHICS 2
```

Zu beachten ist, daß die Cursorposition immer innerhalb des gültigen Zeilenbereichs liegen muß. Hier ein Beispiel:

```
10 GRAPHICS 2
20 FOR I=1 TO 20
30 PRINT #6;"ATARI IST SPITZE"
40 NEXT I
50 GRAPHICS 0
```

Starten Sie dieses Programm, so werden Sie feststellen, daß der Bildschirm zunächst mit dem Text gefüllt wird und der Rechner anschließend mit "*ERROR- 141 AT LINE 30*" alle weiteren Befehle grundsätzlich ablehnt.

ERROR 141 BEDEUTET - CURSORPOSITIONIERUNG UNZULÄSSIG

In der normalen Betriebsart "scrollt" der Bildschirm, d.h. alle Zeilen werden eine Zeile nach oben gerückt, um für die weitere PRINT-Ausgabe Platz zu schaffen. In den Betriebsarten 1 und 2 ist dies nicht möglich. Alle Ausgaben mit 'PRINT #6' müssen demnach unbedingt vom Programm kontrolliert werden.

Dazu können Sie auch in den Betriebsarten 1 und 2 den Befehl "POSITION" einsetzen:

```
10 GRAPHICS 2
20 POSITION 10,4
30 PRINT "A"
40 FOR I=1 TO 1000:NEXT I
50 GRAPHICS 0
```

Hier wird der Buchstabe "A" in der Mitte des Bildschirms ausgegeben und anschließend zur Betriebsart 0 zurückgeschaltet.
Das Textfenster kann ein- und und ausgeblendet werden. Hier ein Beispiel:

```
10 GRAPHICS 2
20 PRINT"ATARI IST SPITZE"
30 FOR I=1 TO 1000:NEXT I
40 GRAPHICS 2+16
50 FOR I=1 TO 1000:NEXT I
60 GRAPHICS 0
```

Das Textfenster ist zwar ausgeblendet worden, jedoch wurde gleichzeitig der Bildschirm gelöscht. Wie kann dies nun vermieden werden?

DER ZUSATZPARAMETER 32 VERHINDERT DAS LÖSCHEN DES BILDSCHIRMS.

Zum Auschalten des Textfensters haben wir zur Grafikbetriebsart die Zahl 16 addiert. Ähnlich ist es hier. Es wird zusätzlich die Zahl '32' addiert, wenn der Bildschirm nicht gelöscht werden soll. Um nach dem letzen Programmbeispiel wieder in die Betriebsart 2 zu gelangen, ohne daß der Bildschirm gelöscht wird, ändern wir die Zeile 40 wie folgt:

```
40 GRAPHICS 2+16+32
```
oder
```
40 GRAPHICS 2+48
```
oder
```
40 GRAPHICS 50
```

Sie sollten die Zahlen 16 und 32 stets zur Betriebsart addieren, um die Übersichtlichkeit des Programms zu gewährleisten. Oder wissen Sie auf Anhieb, was mit 'GRAPHICS 59' gemeint ist?

Im umgekehrten Fall, wenn das Textfenster eingeblendet werden soll, ohne daß der Bildschirm gelöscht wird, wird ebenfalls eine 32 zum GRAPHICS-Befehl addiert:

GRAPHICS 2+32

Dies ist vorerst alles, was Sie zum Befehl GRAPHICS wissen sollten. Im folgenden Abschnitt wenden wir uns den Farben des ATARI zu.

FARBENPRACHT

Wie Sie bereits zur Kenntnis genommen haben, besitzt der ATARI 16 Farben mit je 8 Helligkeitsstufen. Mit diesen Farben kann der Hintergrund, die Schrift, die Grafik und sogar der Rahmen manipuliert werden. Dazu hält der ATARI den folgenden Befehl bereit:

```
Problem:    Farbauswahl

Befehl:     SETCOLOR reg,f,h

Parameter:  reg - Farbregister (0-4)
            f  -  Farbnummer (0-15)
            h  -  Helligkeitsstufe (0-15)

Beispiel:   SETCOLOR 4,3,2
            Ändert die Rahmenfarbe auf Rot
```

Bevor wir diesen Befehl nun einsetzen können, müssen wir die Nummern der 16 Farben kennen. Diese können Sie der folgenden Tabelle entnehmen:

Nummer	Farbe	Nummer	Farbe
0	grau	8	blau
1	goldgelb	9	dunkelblau
2	orange	10	türkis
3	rot	11	grün-blau
4	rosa	12	grün
5	violett	13	gelb-grün
6	flieder	14	orange-grün
7	hellblau	15	orange

Wenn Sie die einzelnen Helligkeitsstufen auch noch kennen, so steht Ihnen wieder ein neuer Befehl zur Verfügung. Es können Werte zwischen 0 und 15 angegeben werden. Dabei ist 0 ganz dunkel und 15 ganz hell. Hier müssen Sie beachten, daß jede gerade Zahl und die darauffolgende Zahl die gleiche Helligkeitsstufe ergibt:

Nummer	Helligkeitsstufe
0,1	1 (dunkel)
2,3	2
4,5	3
6,7	4
8,9	5
10,11	6
12,13	7
14,15	8 (hell)

Die Befehle 'SETCOLOR 4,12,4' und 'SETCOLOR 4,12,5' haben demnach die gleiche Auswirkung.

Aus der Beschreibung des SETCOLOR-Befehls wissen Sie, daß 5 verschiedene Farbregister existieren. In der normalen Grafikbetriebsart (0) befinden sich hier die folgenden Bildschirmfarben:

Register	Inhalt
0	
1	Vordergrundfarbe
2	Hintergrundfarbe
3	
4	Rahmenfarbe

Sie müssen bei der Betriebsart 0 noch darauf achten, daß sich die Vordergrundfarbe, also die Schrift, nicht ändern läßt. Diese Farbe paßt sich stets dem Hintergrund an und kann nur in der Helligkeit geändert werden.

Das folgende Programm stellt den Hintergrund in allen möglichen Farben und Helligkeitsstufen dar:

```
10 FOR FARBE=0 TO 15
20 FOR HELL=0 TO
30 SETCOLOR 2,FARBE,HELL
40 FOR I=1 TO 20:NEXT I
50 NEXT HELL
60 NEXT FARBE
```

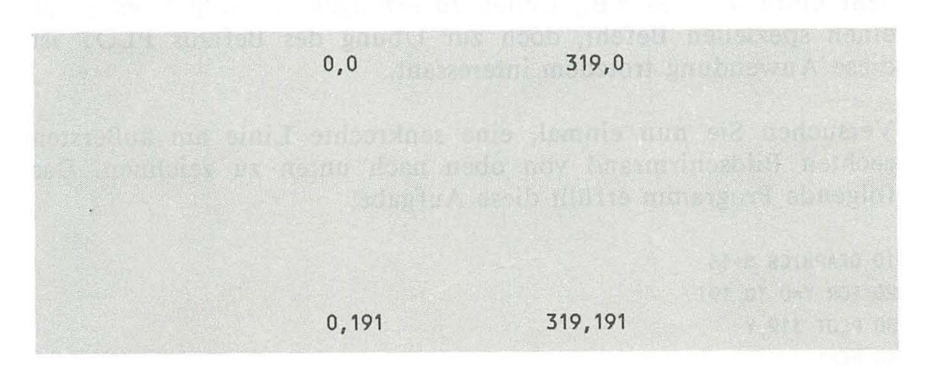
Wenn Sie dieses Programm eingeben und starten, so werden Sie ein eindrucksvolles Schauspiel erleben. Der Bildschirmhintergrund leuchtet nacheinander in allen Farben. Die Farbe wird jeweils von dunkel bis hell variiert.

Die Farbregister sowie die Anzahl der darstellbaren Farben ist bei den Grafikbetriebsarten unterschiedlich. Im folgenden Abschnitt arbeiten wir noch mit den Betriebsarten 8 und 14.

HOCHAUFLÖSENDE GRAFIK

Wie Ihnen bereits bekannt ist, können Sie zwischen 16 verschiedenen Grafikauflösungen mit unterschiedlicher Farbenanzahl wählen. Zum Zeichnen von Figuren gibt es einige Befehle, die ausschließlich am Beispiel der hochauflösenden Grafik vorgestellt werden.

Erinnern Sie sich: Die hochauflösende Grafik besteht aus 320 x 192 Bildpunkten und einer Farbe. Damit jeder Punkt angesprochen werden kann, muß man sich den Bildschirm als Koordinatensystem vorstellen. Ähnlich wie beim bereits bekannten Befehl POSITION, der den Cursor an eine beliebige Position bringen kann, wird auch bei den Grafikbefehlen eine Position bestimmt, nämlich mit zwei Werten. Der erste Wert bestimmt die Position auf der waagerechten Achse (X-Achse) des Bildschirms von links nach rechts orientiert. Der zweite Wert entspricht der Position auf der senkrechten Achse (Y-Achse) von unten nach oben zählend. Die vier Eckpunkte haben folgende Koordinaten:



Wenn Sie sich mit diesem System vertraut gemacht haben, können wir uns mit den ersten Befehlen zur Erzeugung der Grafik beschäftigen.

ZEICHNEN VON PUNKTEN

```
Problem:   Setzen eines Punktes

Befehl:    PLOT x,y

Parameter: x - X-Achsenposition (0-319)
           y - Y-Achsenposition (0-191)

Beispiel:  PLOT 0,0
           Setzt einen Grafikpunkt in die obere,
           linke Ecke des Bildschirms
```

Die Grafikbefehle sind allgemein gut verständlich, wenn man das Koordinatensystem beherrscht. Das Setzen einzelner Punkte wird aber langweilig, wenn man nicht weitere Befehle einsetzt. So ist es sehr interessant, Punkte innerhalb von Schleifen zu erzeugen. Sehr einfach ist es z.B., Linien zu erzeugen. Zwar gibt es dafür einen speziellen Befehl, doch zur Übung des Befehls PLOT ist diese Anwendung trotzdem interessant.

Versuchen Sie nun einmal, eine senkrechte Linie am äußersten rechten Bildschirmrand von oben nach unten zu zeichnen. Das folgende Programm erfüllt diese Aufgabe:

```
10 GRAPHICS 8+16
20 FOR Y=0 TO 191
30 PLOT 319,Y
40 NEXT Y
50 FOR I=1 TO 1000:NEXT I
60 GRAPHICS 0
```

Dieses Programm ist wirklich einfach. Lassen Sie uns den Schwierigkeitsgrad ein wenig erhöhen. Schreiben Sie nun ein Programm, das einen Rahmen um den gesamten Bildschirm erzeugt. Hier die Lösung:

```
10 GRAPHICS 8+16
20 FOR Y=0 TO 191
30 PLOT 0,Y:PLOT 319,Y
40 NEXT Y
50 FOR X=0 TO 319
60 PLOT X,0:PLOT X,191
70 NEXT X
80 FOR I=1 TO 1000:NEXT I
90 GRAPHICS 0
```

Sie haben doch nicht etwa vier Schleifen benötigt? Die beiden parallelen Linien können in einer Schleife gleichzeitig gezogen werden! Doch dies sind kleine Tricks, die die Programmiererfahrung bald mit sich bringt.

ZEICHNEN EINES KREISES

Da der ATARI keinen Befehl zum Zeichnen eines Kreises bereithält, müssen wir uns anders helfen. Jeder einzelne Punkt muß berechnet und gesetzt werden. Dabei benutzen wir die folgenden Formeln:

*X = R * COS(A)*
*Y = R * SIN(A)*

Für 'R' muß der Radius und für 'A' der Winkel eingesetzt werden. In Verbindung mit einer Schleife, die den Winkel von 1 bis 360 Grad hochzählt, kann jeder einzelne Punkt des Kreises erzeugt werden. Beachten Sie zunächst das folgende Programm:

```
10 GRAPHICS 8+16
15 COLOR 3
20 X=160:Y=96:R=30
30 DEG
40 FOR A=1 TO 360
50 PLOT X+R*COS(A),Y+R*SIN(A)
60 NEXT A
70 FOR I=1 TO 1000:NEXT I
80 GRAPHICS 0
```

In der Zeile 20 wird der Mittelpunkt und der Radius des gewünschten Kreises bestimmt. Der Befehl DEG besagt, daß alle folgenden Winkelberechnungen in Neugrad durchzuführen sind. Der alternative Befehl dazu ist RAD und rechnet im Bogenmaß (Altgrad). Dabei gilt '90 Altgrad = 100 Neugrad'. Die folgende Zeile stellt den Schleifenanfang dar. Hier soll ein ganzer Kreis (360 Grad) erzeugt werden. Wenn Sie die Anfangs- und Endwerte ändern, werden Kreisausschnitte erzeugt. In der Zeile 50 ist die eigentliche Formel untergebracht. Zu der errechneten Punktposition muß noch der Mittelpunkt (X und Y) addiert werden. Anschließend folgt das Ende der Schleife.

Mit dieser Routine können Sie nun Kreise erzeugen, auch wenn Sie den mathematischen Zusammenhang nicht verstanden haben. Wenn Sie auch noch Ellipsen erzeugen möchten, so kann ich Ihnen hier nicht weiterhelfen. Vielleicht holen Sie Ihre alten Schulbücher aus dem Schrank und geben sich selbst etwas Nachhilfe in Geometrie. Doch Spaß beiseite - für viele schöne Grafiken sind mathematische Formeln unverzichtbar.

ZEICHNEN VON GERADEN

Problem:	Zeichnen einer Geraden
Befehl:	DRAWTO X,Y
Beispiel:	DRAWTO 319,0 Zeichnet eine Gerade zwischen der Position des Grafikcursors und der angegebenen X/Y - Position

Die einzige Frage, die dieser Befehl aufkommen läßt, ist der Grafikcursor. Zu Anfang, also nach dem Einschalten des Rechners, steht dieser Cursor auf '0,0'. Jeder anschließende Grafikbefehl aktualisiert diesen Cursor. Setzt man also einen Punkt auf '100,150', so steht auch der Grafikcursor auf diesem Punkt. Ein anschließender DRAWTO-Befehl zeichnet die Gerade von diesem Punkt ausgehend. Der letzte Punkt dieser Geraden ist wiederum die neue Position des Grafikcursors.

Zeichnen wir nun eine Senkrechte am äußersten rechten Bildschirmrand, wie wir es zuvor mit dem Befehl PLOT realisiert haben. Dazu benötigen wir nur wenige Befehle:

```
10 GRAPHICS 8+16
20 PLOT 319,0
30 DRAWTO 319,191
40 FOR I=1 TO 1000:NEXT I
50 GRAPHICS 0
```

Nun soll auch dieser Rahmen, der beim Befehl PLOT erzeugt worden ist, mit DRAWTO gezeichnet werden:

```
10 GRAPHICS 8+16
20 FOR Y=0 TO 191 STEP 5
30 PLOT 319,Y
40 DRAWTO 0,191-Y
50 NEXT Y
60 FOR I=1 TO 1000:NEXT I
70 GRAPHICS 0
```

Ein einfaches Programm mit großer Wirkung. Weil's so schön war, folgt noch ein weiteres:

```
10 GRAPHICS 8+16
20 FOR I=0 TO 191 STEP 10
30 DRAWTO 191,I
40 DRAWTO 191-I,191
50 DRAWTO 0,191-I
60 DRAWTO I,0
70 NEXT I
80 FOR I=1 TO 1000:NEXT:GRAPHICS 0
```

Ist es nicht bewundernswert, wenn derartige Bilder auf dem Bildschirm entstehen? Vielleicht versuchen Sie sich auch einmal an derartigen Fantasiegrafiken. Es macht viel Spaß, auch wenn es nicht auf Anhieb klappt. Im Vertrauen, ich habe mir an diesem Bild auch den Kopf zerbrochen. Die Schwierigkeit ist, die Bilder, die man sich vorstellt, in das Koordinatensystem umzurechnen, also den Algorithmus zu bilden. Das artet zu echten Denksportaufgaben aus.

KAPITEL 6: TONERZEUGUNG

Ein Computer eignet sich sehr gut zur Erzeugung von Tönen. Der Prozessor, das eigentliche Arbeitspferd im Computer, wird von einem Taktgenerator angesteuert. Dieser Generator erzeugt je nach Rechner ca. 1-4 MHz (Millionen Takte pro Sekunde). Der einfachste Weg ist, diesen Takt auf einen Lautsprecher zu legen. Da die dabei entstehenden Töne vom menschlichen Gehör nicht mehr wahrgenommen werden können, muß die Taktfrequenz erheblich tiefer sein. Dies wird erreicht, indem die Taktfrequenz vor Ausgabe auf den Lautsprecher geteilt wird. Die dabei entstehenden Frequenzen werden dann als Töne wahrgenommen.

Dies war die erste Generation der Tonerzeugung mit dem Computer. Da diese Töne, vergleichbar mit den Wecktönen einer Armbanduhr, nicht zufriedenstellend waren, wurde die Tonerzeugung in Heimcomputern ständig weiterentwickelt. Das Resultat ist, daß viele Homecomputer mit sogenannten Synthesizern ausgerüstet sind, die auf einem Chip untergebracht sind. Die damit erzeugten Töne sind denen der "natürlichen" Musikinstrumente verblüffend ähnlich.

Den tonerzeugenden Chip im ATARI kann man nicht als Synthesizer im eigentlichen Sinn bezeichnen. Ihm fehlen wichtige Eigenschaften wie verschiedene Wellenformen (Sinus, Rechteck etc.) und die klangbeeinflussenden Filter. Auch ist das eigentliche Prinzip des Synthesizers, die Frequenz (Höhe) der Töne von einer Spannung abhängig zu machen, nicht gegeben. Der Tongenerator im ATARI macht die Frequenz von einem Teilungsfaktor abhängig.

Der Tongenerator des ATARI verfügt über vier Stimmen. Es können also vier Töne gleichzeitig erklingen. Da die Reinheit der Töne bestimmt werden kann, lassen sich auch Geräuscheffekte erzeugen.

Alles, was dem Tongenerator entlockt wird, erklingt über den im Fernsehgerät oder Monitor eingebauten Lautsprecher. Die Lautstärke der Töne kann mit dem ATARI ebenfalls eingestellt werden.

Im folgenden Kapitel lernen Sie die wichtigsten BASIC-Befehle zur Tonerzeugung kennen. Die gesamte, komplexe Tonerzeugung kann an dieser Stelle nicht erläutert werden.

DER SOUND-BEFEHL

```
--------------------------------------------------------
| Problem:   Bestimmen der Toneigenschaften            |
--------------------------------------------------------
| Befehl:    SOUND a,b,c,d                             |
--------------------------------------------------------
| Parameter: a - Stimme (0-3)                          |
|            b - Tonhöhe (0-255)                       |
|            c - Reinheit des Tons (0,2,4,6,8,10,12,14)|
|            d - Lautstärke (0-15)                     |
--------------------------------------------------------
| Beispiel:  SOUND 0,121,10,8,                         |
|            Erzeugt einen Ton mit Stimme 0            |
--------------------------------------------------------
```

Die Parameter des SOUND-Befehls sollen zunächst näher beschrieben werden. Der erste Parameter 'a' gibt die Stimme an. Der nachfolgende Parameter 'b' bestimmt die Tonhöhe. Hier wird ein Wert zwischen 0 und 255 angegeben, wobei 2 der höchste (hörbare) und 255 der tiefste Ton ist. Das folgende Programm läßt einen Ton abschwellen:

```
10 FOR TON=2 TO 255
20 SOUND 0,TON,10,8
30 NEXT TON
40 SOUND 0,0,0,0
```

Die Funktion 'INT' schneidet von dem Wert in den nachfolgenden Klammern die Nachkommastellen ab, da diese in unserem Beispiel nicht benötigt werden. Sie sehen, daß wie bei allen Befehlen auch beim Befehl SOUND alle Parameter mit Variablen besetzt werden können.

Die Veränderung der Tonhöhe in einer Schleife ermöglicht zahlreiche Effekte. So kann z.B. leicht eine Sirene simuliert werden:

```
10 FOR TON=50 TO 100
20 SOUND 0,TON,10,8
30 NEXT TON
40 FOR TON=100 TO 50 STEP -1
50 SOUND 0,TON,10,8
60 NEXT TON
70 GOTO 10
```

Diese Sirene, die an amerikanische Polizeiwagen erinnert, kann mit der Taste 'BREAK' abgebrochen werden. Der letzte Ton erklingt dann allerdings weiter. Um den Ton abzuschalten, geben Sie entweder den Befehl 'SOUND 0,0,0,0' ein oder drücken die RESET-Taste.

Für alle, die noch ein wenig mit der Sirene spielen möchten, bietet sich die folgende Variante des Programms an:

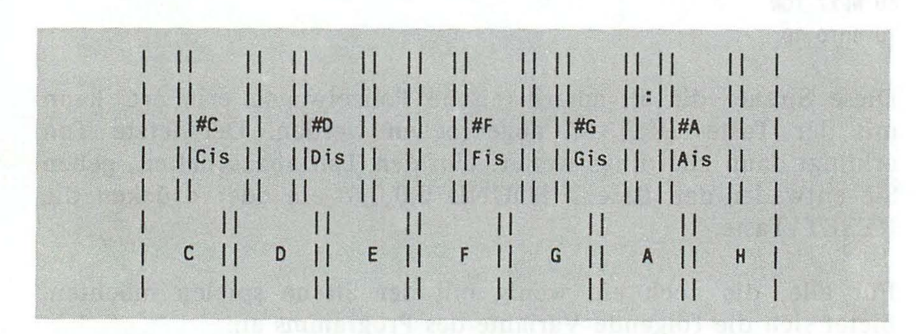
```
5 INPUT "ANFANGSWERT (0-200)";A
6 INPUT "ENDWERT (50-200)   ";B
10 FOR TON=A TO B
20 SOUND 0,TON,10,8
30 NEXT TON
40 FOR TON=B TO A STEP -1
50 SOUND 0,TON,10,8
60 NEXT TON
70 GOTO 10
```

Hier werden die wichtigsten Eigenschaften der gewünschten Sirene zunächst in Variablen festgehalten, die im dann folgenden Programm eingesetzt werden. Auch dieses Programm beenden Sie mit 'BREAK'.

MUSIK AUS FREQUENZEN

Musikinstrumente erzeugen die Töne nicht stufenlos. Erst eine bestimmte Abstufung der Töne klingt für das menschliche Gehör angenehm. Zum weiteren Verständnis müssen einige Grundlagen erläutert werden.

Eine sogenannte Tonleiter besteht aus 12 Tönen, wie Sie der folgenden Übersicht einer Klaviatur entnehmen können.



Eine Klaviatur besteht aus mehreren solcher Abschnitte. Diese Tonfolgen nennt man Oktaven. Die erste Oktave umfaßt die tiefen Töne, die letzte Oktave die höchsten Töne.

Welcher Ton-Parameter entspricht nun welchem Ton in der Tonleiter? Die Antwort finden Sie in der umseitigen Tabelle:

Ton	Oktave 1	Oktave 2	Oktave 3	Oktave 4
C	243	121	60	29
#C	230	114	57	28
D	217	108	53	26
#D	204	102	50	
E	193	96	47	
F	182	91	45	
#F	173	85	42	
G	162	81	40	
#G	153	76	37	
A	144	72	35	
#A	136	68	33	
H	128	64	31	

Wollen Sie also das mittlere 'C' ertönen lassen, so geben Sie den folgenden Befehl ein:

```
SOUND 0,121,10,8
```

Zur Erzeugung einer Tonleiter wären nun 12 SOUND-Befehle nötig. Doch dies geht einfacher. Sie haben bei der Adressenverwaltung bereits Stringtabellen kennengelernt. Solche Tabellen gibt es auch für numerische Werte. Doch diese sind wesentlich einfacher zu programmieren als die Stringtabellen. Angenommen, Sie benötigen eine numerische Tabelle mit 10 Einträgen. Diese Tabelle muß dann am Programmanfang eingerichtet werden. Die folgende Anweisung übernimmt das Reservieren dieser Tabelle:

```
DIM A(10)
```

Dieser Befehl richtet eine Tabelle mit dem Namen 'A' ein, die Platz für 10 Einträge haben soll. Wollen Sie diese Tabelle nun füllen, so geben Sie einfach den Namen der Tabelle und den Tabellenplatz (Index) in Klammern dahinter an. Ein Beispiel:

```
A(3)=1000
```

Hier wird die Zahl 1000 in dem dritten Tabellenplatz gespeichert.

Eine solche Tabelle können wir nun gut für unsere Musikprogramme gebrauchen. Wenn wir mit den ersten drei Oktaven arbeiten wollen, müssen wir für diese Tabelle 36 Plätze reservieren. Wir verwenden den folgenden Befehl, der gleichzeitig den Anfang unserer folgenden Musikprogramme bildet:

```
10 REM ====================
20 REM   LESEN DER NOTEN
30 REM ====================
40 DIM TON(36)
```

Nun müssen wir eine elegante Methode suchen, die 36 Werte in die Tabelle zu bringen. Dazu verwenden wir zwei neue Befehle, die in den folgenden Befehlen eingesetzt werden:

```
50 DATA 243,230,217,204,193,182,173,162,153,144,136,128
60 DATA 121,114,108,102,96,91,85,81,76,72,68,64
70 DATA 60,57,53,50,47,45,42,40,37,35,33,31
80 FOR INDEX=1 TO 36
85 READ A
90 TON(INDEX)=A
95 NEXT INDEX
```

In den DATA-Zeilen werden die Werte, von einem Komma getrennt, festgehalten. Der Befehl READ liest die Daten dann nacheinander in der Schleife. Anschließend wird dieser Wert an den entsprechenden Tabellenplatz gespeichert. Der erste Ton (das tiefste 'C') steht uns nun in TON(1) zur Verfügung. Probieren Sie dies nach dem Start des Programms aus:

```
SOUND 0,TON(1),10,8
```

Mit Hilfe der Tontabelle können wir jetzt ein Programm zum Spielen der Tonleiter erzeugen:

```
10 REM ====================
20 REM   LESEN DER NOTEN
30 REM ====================
40 DIM TON(36)
50 DATA 243,230,217,204,193,182,173,162,153,144,136,128
60 DATA 121,114,108,102,96,91,85,81,76,72,68,64
70 DATA 60,57,53,50,47,45,42,40,37,35,33,31
80 FOR INDEX=1 TO 36
85 READ A
90 TON(INDEX)=A
95 NEXT INDEX
100 REM ====================
110 REM     TONLEITER
120 REM ====================
130 PRINT "OKTAVE  (1-3)";
140 INPUT O
150 FOR T=(O-1)*12+1 TO O*12
160 SOUND 0,TON(T),10,8
170 FOR I=1 TO 50:NEXT I
180 SOUND 0,0,0,0
190 NEXT T
```

Es wird immer interessanter, mit dem Tongenerator zu experimentieren. Nun werden wir diesem Kapitel die Krone aufsetzen. Das folgende Programm "KLAVIER FÜR EINSTEIGER" verwandelt Ihren ATARI in ein Musikinstrument.

```
10 REM ====================
20 REM   LESEN DER NOTEN
30 REM ====================
40 DIM TON(36)
50 DATA 243,230,217,204,193,182,173,162,153,144,136,128
60 DATA 121,114,108,102,96,91,85,81,76,72,68,64
70 DATA 60,57,53,50,47,45,42,40,37,35,33,31
80 FOR INDEX=1 TO 36
85 READ A
90 TON(INDEX)=A
95 NEXT INDEX
```

```
100 REM ====================
110 REM      KLAVIER
120 REM ====================
130 DIM TAST$(17):DIM T1(255)
135 FOR I=0 TO 255:T1(I)=0:NEXT I
140 TAST$="Q2W3ER5T6Y7UI9O0P"
150 FOR I=1 TO 17
160 T1(ASC(TAST$(I,I)))=I
170 NEXT I
180 GRAPHICS 2+16
190 PRINT #6;"    KLAVIER FUER"
200 PRINT #6;"     EINSTEIGER "
210 PRINT #6:PRINT #6
220 PRINT #6;" 2 3  5 6 7  9 0"
230 PRINT #6;" Q W E R T Y U I O P"
240 OPEN #1,4,0,"K:"
250 GET #1,A
260 IF T1(A)<=0 THEN 250
270 T=T1(A)
280 FOR D=15 TO 0 STEP -1
290 SOUND 0,TON(T+12),10,D
300 NEXT D
310 GOTO 250
```

Ein kleines Programm mit großer Wirkung! Vielleicht nehmen Sie noch einige Verbesserungen vor, nachdem Sie die Arbeitsweise durchschaut haben.

GERÄUSCHE

Der SOUND-Befehl bietet mit seinem dritten Parameter die Möglichkeit, den Ton zu verzerren. Wird hier eine 10 oder eine 14 angegeben, so erklingt ein sauberer Ton. Doch was geschieht, wenn hier ein anderer Wert gesetzt wird?

Wird hier eine andere gerade Zahl (0, 2, 4,6, 8 oder 12) angegeben, so wird der Ton mehr oder weniger verzerrt. Dabei können unter Umständen Töne erzeugt werden, die tiefer oder höher als der entsprechende "saubere" Ton sind. Die beiden folgenden, fast identischen SOUND-Befehle bestätigen dies:

```
SOUND 0,121,10,8
SOUND 0,121,12,8
```

Geben Sie für die Verzerrung einen ungeraden Wert ein, so wird die Stimme abgeschaltet.

Das folgende Programm demonstriert alle möglichen Variationen zwischen Tonhöhe und Verzerrung:

```
10 REM ====================
20 REM    VERZERRUNGEN 1
30 REM ====================
40 FOR T=2 TO 255
50 FOR V=0 TO 14 STEP 2
60 PRINT T,V
70 SOUND 0,T,V,8
80 FOR I=1 TO 50:NEXT I
90 NEXT V
100 NEXT T
```

Wenn Ihnen diese Demonstration nicht so gut gefällt, so können Sie mit der Taste 'BREAK' oder 'RESET' abbrechen.

Wenn Sie das gesamte Frequenzspektrum einer Verzerrung testen möchten, so hilft Ihnen das folgende Programm weiter:

```
10 REM ====================
20 REM    VERZERRUNGEN 2
30 REM ====================
40 PRINT "VERZERRUNG (0-15) ";
50 INPUT V
60 FOR T=2 TO 255
70 SOUND 0,T,V,8
80 NEXT T
```

Zum Abschluß dieses Kapitels sollen Sie die folgenden Geräuschprogramme zu eigenen Experimenten anregen:

```
10 REM ====================
20 REM    EXPLOSION
30 REM ====================
40 FOR L=15 TO 0 STEP -0.03
50 SOUND 0,50,0,L
60 NEXT L
```

```
10 REM ====================
20 REM  VOGELGEZWITSCHER
30 REM ====================
40 FOR T=4 TO 14
50 SOUND 0,T,10,8
60 NEXT T
70 A=RND(0)*50
80 FOR I=1 TO A
90 GOTO 40
```

```
10 REM ====================
20 REM    TRUCKER-HUPE
30 REM ====================
40 SOUND 0,253,10,8
50 SOUND 1,220,10,8
60 SOUND 2,8,2,2
70 FOR I=1 TO 1000:NEXT I
80 FOR A=0 TO 3
90 SOUND I,0,0,0
100 NEXT A
```

KAPITEL 7: NOCH MEHR BEFEHLE

Wenn Sie dieses Kapitel des Buches erfolgreich erreicht haben, so werden Sie es sicher nicht erwarten können, weitere Befehle des ATARI-BASIC kennenzulernen. Dem soll nichts im Wege stehen. Sie werden nun mit Befehlen konfrontiert, deren Komplexität sich in Grenzen hält. Zu allen Befehlen finden Sie wie gewohnt eine ausführliche Beschreibung sowie erklärende Beispielprogramme.

JOYSTICKABFRAGE

Wenn Sie Ihren ATARI ausnahmsweise von der Seite betrachten, werden Sie zwei Anschlüsse bemerken, die an fast jedem Homecomputer zu finden sind. Hier werden die sogenannten Joysticks angeschlossen, die hauptsächlich dazu dienen, Spiele optimal zu steuern.

Da die Abfrage der Stellung eines Joysticks beim ATARI recht einfach ist, werden wir den entsprechenden Befehl nun kennenlernen.

```
Problem:    Abfrage des Joysticks

Funktion:   STICK (n)

Parameter:  n - Nummer des Joysticks (0 oder 1)

Beispiel:   PRINT STICK(0)
            Zeigt den Code der Joystickbetätigung auf
            dem Bildschirm.
```

```
Bemerkung: Es gelten folgende Codes:

                  14
              10       6      keine Betätigung=15
                 . . .
              11 . . . 7
                 . . .
               9       5
                  13
```

Schreiben wir nun zunächst ein Programm, das die Betätigung des Joysticks optisch am Bildschirm zu erkennen gibt.

```
10 PRINT STICK(0)
20 GOTO 10
```

Wenn Sie dieses Programm eingeben und starten, so werden Sie die Codes jeder Betätigung des Joysticks erkennen. Auf diese Werte kann das entsprechende Programm dann reagieren.

Nun fehlt nur noch die Abfrage des Feuerknopfes. Dazu gibt es im ATARI-BASIC den folgenden Befehl:

```
Problem:    Abfrage des Feuerknopfes

Funktion:   STRIG (n)

Parameter:  n - Nummer des Joysticks (0 oder 1)

Beispiel:   PRINT STICK(n)
            Ergibt eine 0 bei gedrücktem und eine 1
            bei nicht gedrücktem Feuerknopf
```

Auch hier wieder ein Beispielprogramm. Schließen Sie den Joystick 0 (links) an und starten das folgende Programm.

```
10 A=STRIG(0)
20 IF A=1 THEN 10
30 SOUND 0,200,2,8
40 FOR I=1 TO 100:NEXT I
50 GOTO 10
```

Dieses Programm erzeugt bei gedrücktem Feuerknopf ein Geräusch.

Mit diesen beiden Funktionen läßt sich ein eindrucksvolles Programm zum Zeichnen auf dem Bildschirm schreiben. Der Feuerknopf soll zum Löschen des Bildschirms benutzt werden.

```
10 GRAPHICS 8+16
20 X=160:Y=96
30 PLOT X,Y
40 A=STICK(0)
50 B=STRIG(0)
60 IF A=7 AND X<319 THEN X=X+1:GOTO 30
70 IF A=11 AND X>0 THEN X=X-1:GOTO 30
80 IF A=13 AND Y<191 THEN y=y+1:GOTO 30
90 IF A=14 AND Y>0 THEN Y=Y-1:GOTO 30
100 IF B=0 THEN GRAPHICS 8+16
110 GOTO 40
```

Dieses Programm ist recht einfach gestaltet. Sicher werden Sie keine Probleme haben, die Arbeitsweise zu verstehen und kleine Verbesserungen vorzunehmen. So wäre z.B. eine Erweiterung auf diagonales Zeichnen sehr interessant. Ein Tip: Sie benötigen dazu vier weitere IF-Abfragen, die jeweils beide Variablen ('X' und 'Y') auf einen gültigen Wert überprüfen und aktualisieren müssen.

ZUFALLSZAHLEN

Fast jeder BASIC-Anfänger schreibt seine ersten Spiele mit Hilfe von Zufallszahlen. Ohne Zufallszahlen wäre die Programmierung von Glücksspielen unmöglich. Der ATARI erzeugt mit einer Funktion Zufallszahlen zwischen 0.000000001 und 0.999999999:

```
Problem:    Erzeugen von Zufallszahlen

Funktion:   RND(0)

Beispiel:   PRINT RND(0)
            Erzeugt eine Zufallszahl und gibt diese
            auf dem Bildschirm aus.

Bemerkung:  Die erzeugte Zahl liegt zwischen
            0.000000001 und 0.999999999
```

Ein Befehl, der leicht anzuwenden ist. Schwieriger wird es, wenn die Zahl innerhalb eines anderen Bereichs liegen soll. Dazu muß der Zufallswert aufbereitet werden. Soll z.B. eine Zahl zwischen 0 und 999 ermittelt werden, so muß die Zufallszahl wie im folgenden Programm aufbereitet werden:

```
10 REM ****************
20 REM    WUERFEL
30 REM ****************
40 ZAHL=RND(0)
50 PRINT INT(ZAHL*1000)
```

Die Zufallszahl wird also mit 1000 multipliziert. Der Bereich der möglichen Zufallszahlen liegt dann zwischen 0.000001 und 999.999999. Da die Nachkommastellen nicht benötigt werden, werden sie mit der Funktion INT "abgeschnitten".

Eine weitere Schwierigkeit entsteht, wenn die gewünschte Zahl nicht bei Null, sondern bei einer anderen Zahl beginnen soll. Mit folgender Formel kann jeder beliebige Bereich von Zufallszahlen erzeugt werden:

```
Umrechnen von Zufallszahlen

INT (RND(0) * ((B+1)-A))+A

A - erste gültige Zahl
B - letzte gültige Zahl
```

Sollen beispielsweise Zahlen zwischen 1 und 6 ermittelt werden, so wird die erforderliche Formel wie folgt aufgebaut:

*INT(RND(0) * ((6-1)+1))+1*

ergibt

*INT(RND(0) * (5+1))+1*

ergibt

*INT(RND(0) * 6)+1*

Das abschließende Programm soll Lottozahlen (6 aus 49) erzeugen:

```
10 REM ***************
20 REM   LOTTOZAHLEN
30 REM ***************
40 PRINT "WIEVIELE REIHEN ";REIHEN
50 FOR I1=1 TO REIHEN
60 FOR I2=1 TO 6
70 PRINT INT(RND(0)*49)+1;" - ";
80 NEXT I2
90 PRINT
100 NEXT I1
```

Auch dieses Programm werden Sie schnell durchschauen. Es hat aber einen Haken: In einer Reihe können zwei oder mehrere gleiche Zahlen auftauchen. Dies zu vermeiden, soll nun Ihre Aufgabe sein. Ein Tip: Speichern Sie alle in einer Reihe bereits gezogenen Zahlen in einer Tabelle, die Sie dann mit der aktuellen Zahl vergleichen. Nur wenn die ermittelte Zahl nicht in der Tabelle enthalten ist, wird sie akzeptiert. Vor der nächsten Reihe muß diese Tabelle aber wieder gelöscht werden. Viel Spaß!

ANHANG A: ADRESSEN AUF DISKETTE

Unser Adressenverwaltungs-Programm ist ausschließlich zur Speicherung auf einer Kassette ausgelegt. Für alle diejenigen, die ein ATARI-Diskettenlaufwerk besitzen, ist dieser Anhang bestimmt. Es werden die beiden Unterprogramme zum Speichern und Lesen der Daten gezeigt, die sich nur geringfügig von den bestehenden Routinen unterscheiden. Es muß lediglich die Gerätebezeichnung von "C:" in "D:" umgewandelt werden. Weiterhin ist die Anweisung zur Vorbereitung entsprechend angepaßt worden. Ändern Sie also nur diese Zeilen ab.

```
5000 REM ====================
5010 REM    DATEI LADEN
5020 REM ====================
5040 POSITION 2,10
5050 PRINT "FOLGENDE VORBEREITUNGEN DURCHFUEHREN:"
5060 PRINT "-------------------------------------"
5070 PRINT
5080 PRINT "1. DISKETTE ZUM LADEN EINLEGEN"
5090 PRINT "2. TASTE RETURN BETAETIGEN"
5100 INPUT ANTWORT$
5180 OPEN #1,4,0,"D:ADRESSEN"
5190 INPUT #1;Z
5200 FOR ZEIGER=1 TO Z
5210 FOR INDEX=1 TO 7
5220 GOSUB 600
5225 INPUT #1;ZW$
5230 TABELLE$(VON,BIS)=ZW$
5240 NEXT INDEX
5250 NEXT ZEIGER
5260 CLOSE #1
5270 GOSUB 300
5280 PRINT "DIE ADRESSEN SIND GELADEN!"
5290 FOR I=1 TO 1000:NEXT I
5300 GOTO 1000
```

```
10000 REM ====================
10010 REM    DATEI SPEICHERN
10020 REM ====================
10030 GOSUB 500
10040 POSITION 2,10
10050 PRINT "FOLGENDE VORBEREITUNGEN DURCHFUEHREN:"
10060 PRINT "-------------------------------------"
10070 PRINT
10080 PRINT "1. DISKETTE ZUM SPEICHERN EINLEGEN"
10090 PRINT "2. TASTE RETURN BETAETIGEN"
10100 INPUT ANTWORT$
10180 OPEN #1,8,0,"D:ADRESSEN"
10190 PRINT #1;ZEIGER
10195 Z=ZEIGER
10200 FOR ZEIGER=1 TO Z
10210 FOR INDEX=1 TO 7
10220 GOSUB 600
10230 PRINT #1;TABELLE$(VON,BIS)
10240 NEXT INDEX
10250 NEXT ZEIGER
10260 CLOSE #1
10270 GOSUB 300
10280 PRINT "DIE ADRESSEN SIND GESPEICHERT!"
10290 FOR I=1 TO 1000:NEXT I
10300 GOTO 1000
```

ANHANG B: ABKÜRZUNGEN FÜR BEFEHLE UND FUNKTIONEN

Sie lernen nun eine vereinfachte Methode der Befehlseingabe kennen. Diese Möglichkeit wurde absichtlich nicht vor der ersten Programmerstellung angeboten, da Sie die Befehle und nicht die Abkürzungen lernen sollten.

Eine Befehlsabkürzung haben Sie bereits kennengelernt, das Fragezeichen für PRINT. Der Befehl PRINT hat noch eine weitere Abkürzung. Sie besteht aus den Buchstaben 'PR' und einem abschließenden Punkt. Geben Sie doch einmal folgendes Beispiel ein:

PR. "VEREINFACHTER PRINT"

Sie sehen, daß diese Abkürzung die gleiche Funktion wie der ausgeschriebene PRINT-Befehl hat. Dies ist auch bei allen anderen Befehlen und Funktionen der Fall.

Die folgenden Befehle und Funktionen dürfen auf keinen Fall als Variablennamen beutzt werden, da sonst eine Fehlermeldung erscheint.

Die folgende Übersicht zeigt Ihnen die Abkürzungen aller Befehle und Funktionen

BEFEHL	ABKÜRZUNG	BEFEHL	ABKÜRZUNG
ABS		NEXT	N.
ADR		NOT	
AND		NOTE	NO.
ASC		ON	
ATN		OPEN	O.
BYE	B.	OR	
CLOAD	CLOA.	PADDLE	
CHR$		PEEK	

BEFEHL	ABKÜRZUNG	BEFEHL	ABKÜRZUNG
CLOG		PLOT	PL.
CLOSE	CL.	POINT	P.
CLR		POKE	POK.
COLOR	C.	POP	
COM		POSITION	POS.
CONT		PRINT	PR. ODER ?
COS		PTRIG	
CSAVE	CS.	PUT	PU.
DATA		RAD	
DEG	DE.	READ	REA.
DIM	DI.	REM	R.
DOS	DO.	RESTORE	RES.
DRAWTO	DR.	RETURN	RET.
END		RND	
ENTER	E.	RUN	RU.
EXP		SAVE	S.
FOR	F.	SETCOLOR	SE.
FRE		SGN	
GET	GE.	SIN	
GOSUB	GOS.	SOUND	SO.
GOTO	G.	SQR	
GRAPHICS	GR.	STATUS	ST.
IF		STEP	
INPUT	I.	STICK	
INT		STRIG	
LEN		STOP	STO.
LET	LE.	STR$	
LIST	L.	THEN	
LOAD	LO.	TO	
LOCATE	LOC.	TRAP	T.
LOG		USR	
LPRINT	LP.	VAL	
NEW		XIO	X.

ANHANG C: FEHLERMELDUNGEN

Da der ATARI die Fehlermeldungen numeriert hat und bei Auftritt eines Fehlers auch nur diese Nummer ausgibt, folgt nun eine Liste der Fehlermeldungen mit den entsprechenden Nummern:

2	**Speicherkapazität überschritten**
3	**Zahlenbereich falsch**
4	**Mehr als 128 Variablen**
5	**String ist zu lang**
6	**Zu wenig Daten (beim Einlesen mit READ)**
7	**Zahl größer als 32767**
8	**Falscher INPUT-Befehl**
9	**DIM-Fehler**
10	**Nicht ausführbar**
11	**Zahlenbereich verlassen**
12	**Zeile nicht gefunden**
13	**Kein entsprechender FOR-Befehl**
14	**Zeile zu lang**
15	**GOSUB/FOR entfernt**
16	**RETURN ohne entsprechendes GOSUB**
17	**Syntax-Fehler**
18	**Ungültiges String-Zeichen**
19	**Programm zu lang**
20	**Gerätenummer zu groß**
21	**Data-Fehler**

Folgende Fehlermeldungen treten bei angeschlossenen Zusatzgeräten auf:

128	**mit BREAK abgebrochen**
129	**IOCB schon geöffnet**
130	**Gerät ist unbekannt**
131	**IOCB, nur Ausgabe möglich**
132	**ungültiger "HANDLER" -Befehl**
133	**Gerät oder Datei nicht vorbereitet (OPEN)**
134	**ungültige IOCB-Nummer**
135	**IOCB, nur Eingabe möglich**
136	**Datei-Ende mit EOF**
137	**Datensatz verstümmelt**
138	**Gerät antwortet nicht**
139	**Gerät arbeitet nicht einwandfrei**
140	**Fehler auf dem seriellen Bus**
141	**Cursor außerhalb der Bildschirmbegrenzung**
142	**Formatfehler bei Datenübertragung**
143	**Prüfsummenfehler bei Datenübertragung**
144	**Diskettenfehler**
145	**Diskettenschreibfehler**
146	**Funktion nicht ausführbar**
147	**Grafikspeicher reicht nicht aus**
160	**Unbekannt Diskettenstation**
161	**Zu viele Files offen**
162	**Diskettenkapazität erreicht**
163	**Systemfehler**
164	**File/Sektor-Fehler**
165	**Falscher Filename**
166	**POINT-Befehl fehlerhaft**
167	**FILE ist schreibgeschützt**
168	**unbekannter XIO-Befehl**
169	**Kein Platz in der Directory**
170	**File nicht gefunden**
171	**POINT-Befehl nicht ausführbar**
172	**Unzulässiger Anhang**
173	**falsches Format**

ANHANG D:
INTERESSANTE SPEICHERSTELLEN

Sie haben bereits den Befehl POKE kennengelernt, mit dem Speicherstellen im ATARI direkt verändert werden können. Ein Beispiel ist die Einschaltung des internationalen Zeichensatzes. Dies geschieht mit dem Befehl 'POKE 756,204'. In dieser Speicherstelle 756 ist also festgehalten, welcher Zeichensatz verwendet wird.

Es besteht weiterhin die Möglichkeit, Speicherstellen abzufragen. Die entsprechende Funktion dazu ist 'PEEK(n)'. Für den Parameter 'n' wird die gewünschte Speicherstelle eingesetzt. Im folgenden Beispiel soll der eingeschaltete Zeichensatz abgefragt werden:

PRINT PEEK(756)

Ist der normale Zeichensatz eingeschaltet, so ergibt dieser Befehl den Wert 224.

Die folgende Aufstellung zeigt weitere, interessante Speicherstellen und verdeutlicht ihre Handhabung an einem Beispiel:

BILDSCHIRM

77 FARBWECHSEL

Sie haben sicher schon bemerkt, daß Ihr ATARI selbständig die Bildschirmfarben wechselt, wenn länger als 9 Minuten keine Taste betätigt wird. Diese Speicherstelle enthält eine 0, wenn kein Farbwechsel vorliegt, und eine 254, wenn länger als 9 Minuten keine Taste betätigt wurde.

82 LINKER BILDSCHIRMRAND

Nach dem Einschalten ist der linke Bildschirmrand auf Spalte 2 eingestellt. Die Zeilen können aus diesem Grund nur 38 Zeichen enthalten. Wollen Sie diesen Rand ändern, so setzen Sie in diese Speicherstelle einen anderen Wert. Ein Beispiel: Der linke Bildschirmrand soll auf 0 gestellt werden, damit in einer Zeile 40 Zeichen dargestellt werden können:

```
POKE 82,0
```

83 RECHTER BILDSCHIRMRAND

Diese Speicherstelle entspricht der zuvor aufgeführten. Mit Hilfe der beiden Speicherstellen 82 und 83 kann nun z.B. der Bildschirm so eingestellt werden, daß nur die 20 Zeichen in der Mitte des Bildschirms genutzt werden können:

```
POKE 82,20:POKE 83,29
```

84 AKTUELLE CURSORPOSITION (ZEILE)

Diese Speicherstelle enthält die momentane Zeilenposition des Cursors. Beispiel: Die aktuelle Cursor-Zeilenposition soll in der Variablen 'CZ' gespeichert werden:

```
CZ=PEEK(84)
```

85 AKTUELLE CURSORPOSITION (SPALTE)

Hier ist die Spaltenosition des Cursors enthalten. Sowohl die Zeilen- als auch die Spaltenposition kann nicht nur gelesen, sondern auch geschrieben werden. Beispiel:

```
POKE 84,12:POKE 85,18:PRINT "MITTE"
```

87 GRAFIKBETRIEBSART

Diese Speicherstelle enthält die eingeschaltete Grafikbetriebsart. Beispiel: Die Grafikbetriebsart soll in der Variablen 'GB' gespeichert werden:

```
GB=PEEK(87)
```

93 ZEICHEN UNTER CURSOR

Hier ist das Zeichen gespeichert, das sich unter dem Cursor befindet.

752 CURSOR EIN/AUS

Diese Speicherstelle enthält normalerweise den Wert 0. Wird hier eine 1 gespeichert, so wird der Cursor ausgeschaltet. Beispiel:

```
POKE 752,1
```

755 CURSOR- UND ZEICHENMANIPULATION

Diese Speicherstelle ist sehr interessant. Hier kann der Cursor durchsichtig oder undurchsichtig, sichtbar oder unsichtbar und die Zeichen normal oder auf dem Kopf dargestellt werden. Durch diese Vielzahl von Möglichkeiten können verschiedene Werte in diese Speicherstelle gesetzt werden. Die folgende Tabelle gibt darüber Aufschluß:

INHALT	CURSOR		ZEICHENDARSTELLUNG	
755	DURCHS.	UNDURCHS.	NORMAL	AUF KOPF
0	X		X	
1		X	X	
2	X		X	
3		X	X	
4	X			X
5		X		X
6	X			X
7		X		X

Ein Beispiel: Die Zeichen sollen auf dem Kopf stehend, der Cursor durchsichtig dargestellt werden:

POKE 755,6

756 ZEICHENSATZAUSWAHL

Diese Speicherstelle enthält 224, wenn der normale, und 204, wenn der internationale Zeichensatz eingeschaltet ist. Beispiel: Es soll der internationale Zeichensatz eingeschaltet werden:

```
POKE 756,204
```

TASTATUR

17 BREAK-TASTE

Diese Speicherstelle enthält eine 0, wenn die BREAK-TASTE gedrückt wurde, sonst ist hier eine 128 gespeichert.

694 ZEICHENCODIERUNG

In dieser Speicherstelle befindet sich der Wert, der zu dem Code des gedrückten Zeichens addiert wird. Da alle reversen Zeichen um 128 größer sind als die entsprechenden normalen Zeichen, so kann mit einem 'POKE 694,128' auf reverse Schrift umgeschaltet werden. Standardmäßig befindet sich hier eine 0. Beispiel: Es soll jewiels das nächste des gedrückten Zeichens erscheinen (Code+1):

```
POKE 694,1
```

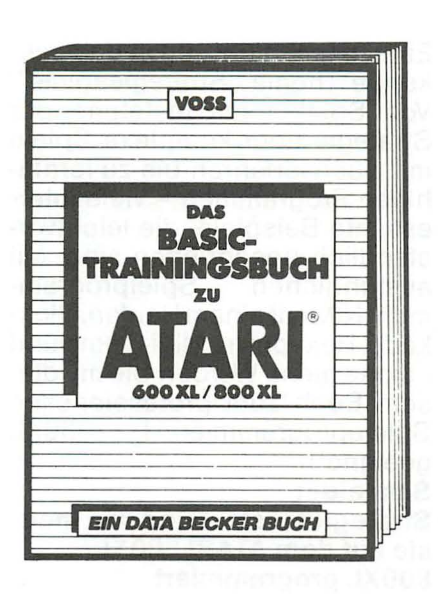
53279 FUNKTIONSTASTEN

Über diese Speicherstelle kann die betätigte Funktionstatse abgefragt werden. Folgende Tabelle gilt dabei:

INHALT 53279	betätigte Funktionstasten OPTION	SELECT	START
0	X	X	X
1	X	X	
2	X		X
3	X		
4		X	X
5		X	
6			X
7			

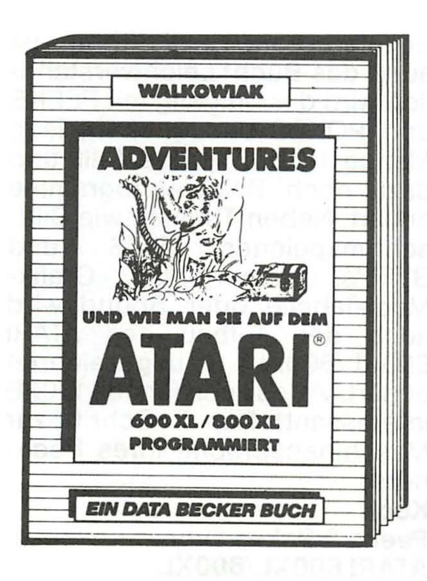
Folgendes Programm zeigt den entsprechenden Wert von 53279:

```
10 IF PEEK(53279)=7 THEN 10
20 PRINT PEEK(53279)
30 GOTO 10
```



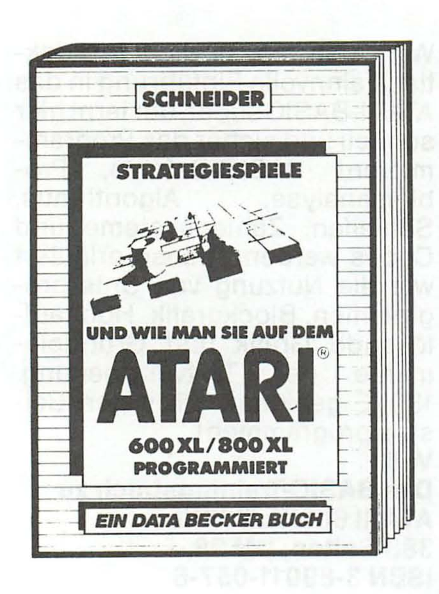
Wer eine ausführliche, didaktisch sinnvolle Einführung in das ATARI-BASIC sucht, der lernt hier schnell und sicher das Programmieren! BASIC-Befehle, Problemanalyse, Algorithmus, Schleifen, Zahlensysteme und Codes werden ebenso erläutert wie die Nutzung von Unterprogrammen, Blockgrafik, Hochauflösende Grafik und Grundelemente der Textverarbeitung. 130XE geeignet! Mit vielen Beispielprogrammen!

Voß
Das BASIC-Trainingsbuch zu ATARI 600XL/800XL
383 Seiten, DM 39,–
ISBN 3-89011-057-6



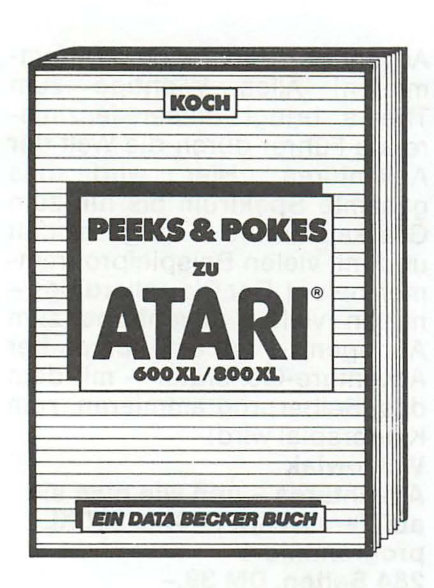
Adventures erfolgreich programmieren! Alles Wichtige zum Thema bringt dieser faszinierende Führer durch die Welt der Adventures. Hier wird das gesamte Spektrum bis hin zum Grafikadventure abgehandelt und mit vielen Beispielprogrammen belegt. Der Clou allerdings – neben vielen Adventures zum Abtippen – ist ein kompletter Adventure-Generator – mit dem das Selberprogrammieren zum Kinderspiel wird!

Walkowiak
Adventures – und wie man sie auf dem ATARI 600XL/800XL programmiert
284 Seiten, DM 39,–
ISBN 3-89011-059-2



Eine tolle Einführung in das pakkende Thema „Strategiespiele"! Von Spielen mit feststehender Strategie über komplexe Spiele mit Suchverfahren bis zu lernfähigen Programmen – viele interessante Beispiele, die leichtverständlich beschrieben sind. Mit ausführlichen Spielprogrammen: NIM mit einem Haufen, Blokkade, Hexapawn, Mini-Dame und etliche mehr. Werden Sie mit diesem Buch zum professionellen Spieleprogrammierer! 130XE geeignet!

Schneider
Strategiespiele – und wie man sie auf dem ATARI 600XL/ 800XL programmiert
181 Seiten, DM 29,–
ISBN 3-89011-077-0



So interessant wie das Thema ist auch das Buch! Leichtverständlich wird der Umgang mit PEEKS und POKES beschrieben, jede Menge POKES dargestellt und dazu noch Beispielprogramme erklärt. Neben Themen wie Bildschirmspeicher, BITS und BYTE's, Memory-Map, Grafik-Modi-Tabelle oder Sound wird auch der Aufbau des ATARI 600XL/800XL ausgezeichnet erklärt. Vieles auch für den 130XE interessant! Der 1. Schritt zur Maschinensprache Ihres Rechners!

Koch
Peeks & Pokes zum ATARI 600XL/800XL
251 Seiten, DM 39,–
ISBN 3-89011-082-7

DAS STEHT DRIN:

Das Buch ATARI für Einsteiger sollte das erste Buch zum ATARI 130 XE, 600 XL oder 800 XL sein. Es ist eine leichtverständliche Einführung in Handhabung, Einsatz und Programmierung des ATARI-Homecomputers, die keinerlei Vorkenntnisse voraussetzt.

Aus dem Inhalt:

- Die Bedienung der Tastatur und des Editors
- Der erste Befehl
- Das erste Programm
- BASIC-Einführung Schritt für Schritt mit Erstellung einer kompletten Adressenverwaltung
- Grafikbefehle mit Beispielprogrammen
- Sound-Befehle mit eindrucksvollen Demonstrationsprogrammen (z.B. Orgel auf der Tastatur)
- Das Kassettenlaufwerk und seine Bedienung
- Weitere nützliche Befehle (z.B. Joystickabfrage, Zufallszahlen)

UND GESCHRIEBEN HAT DIESES BUCH:

Norbert Szczepanowski, EDV-Kaufmann, ist Bestsellerautor bei DATA BECKER. Er hat mehrjährige Erfahrung in der Programmierung zahlreicher Rechner.

ISBN 3-89011-033-9